# W3School jQuery 参考手册

wizardforcel

Published with GitBook



# 目錄

# W3School jQuery 参考手册

来源：jQuery 参考手册

整理：飞龙

# jQuery 参考手册 - 选择器

## jQuery 选择器

| 选择器 | 实例 | 选取 |
|---|---|---|
| * | $("*") | 所有元素 |
| *#id* | $("#lastname") | id="lastname" 的元素 |
| *.class* | $(".intro") | 所有 class="intro" 的元素 |
| *element* | $("p") | 所有 <p> 元素 |
| *.class.class* | $(".intro.demo") | 所有 class="intro" 且 class="demo" 的元素 |
| :first | $("p:first") | 第一个 <p> 元素 |
| :last | $("p:last") | 最后一个 <p> 元素 |
| :even | $("tr:even") | 所有偶数 <tr> 元素 |
| :odd | $("tr:odd") | 所有奇数 <tr> 元素 |
| :eq(*index*) | $("ul li:eq(3)") | 列表中的第四个元素（index 从 0 开始） |
| :gt(*no*) | $("ul li:gt(3)") | 列出 index 大于 3 的元素 |
| :lt(*no*) | $("ul li:lt(3)") | 列出 index 小于 3 的元素 |
| :not(*selector*) | $("input:not(:empty)") | 所有不为空的 input 元素 |
| :header | $(":header") | 所有标题元素 <h1> - <h6> |
| :animated | | 所有动画元素 |
| :contains(*text*) | $(":contains('W3School')") | 包含指定字符串的所有元素 |
| :empty | $(":empty") | 无子（元素）节点的所有元素 |
| :hidden | $("p:hidden") | 所有隐藏的 <p> 元素 |
| :visible | $("table:visible") | 所有可见的表格 |
| s1,s2,s3 | $("th,td,.intro") | 所有带有匹配选择的元素 |
| [*attribute*] | $("[href]") | 所有带有 href 属性的元素 |
| [*attribute*=*value*] | $("[href='#']") | 所有 href 属性的值等于 "#" 的元素 |
| [*attribute*!=*value*] | $("[href!='#']") | 所有 href 属性的值不等于 "#" 的元素 |
| [*attribute*$=*value*] | $("[href$='.jpg']") | 所有 href 属性的值包含以 ".jpg" 结尾的元素 |

| | | |
|---|---|---|
| :input | $(":input") | 所有 <input> 元素 |
| :text | $(":text") | 所有 type="text" 的 <input> 元素 |
| :password | $(":password") | 所有 type="password" 的 <input> 元素 |
| :radio | $(":radio") | 所有 type="radio" 的 <input> 元素 |
| :checkbox | $(":checkbox") | 所有 type="checkbox" 的 <input> 元素 |
| :submit | $(":submit") | 所有 type="submit" 的 <input> 元素 |
| :reset | $(":reset") | 所有 type="reset" 的 <input> 元素 |
| :button | $(":button") | 所有 type="button" 的 <input> 元素 |
| :image | $(":image") | 所有 type="image" 的 <input> 元素 |
| :file | $(":file") | 所有 type="file" 的 <input> 元素 |
| :enabled | $(":enabled") | 所有激活的 input 元素 |
| :disabled | $(":disabled") | 所有禁用的 input 元素 |
| :selected | $(":selected") | 所有被选取的 input 元素 |
| :checked | $(":checked") | 所有被选中的 input 元素 |

## 参阅

教程：jQuery 元素选择器语法

# jQuery 参考手册 - 事件

## jQuery 事件方法

事件方法会触发匹配元素的事件，或将函数绑定到所有匹配元素的某个事件。

触发实例：

```
$("button#demo").click()
```

上面的例子将触发 id="demo" 的 button 元素的 click 事件。

绑定实例：

```
$("button#demo").click(function(){$("img").hide()})
```

上面的例子会在点击 id="demo" 的按钮时隐藏所有图像。

| 方法 | 描述 |
| --- | --- |
| bind() | 向匹配元素附加一个或更多事件处理器 |
| blur() | 触发、或将函数绑定到指定元素的 blur 事件 |
| change() | 触发、或将函数绑定到指定元素的 change 事件 |
| click() | 触发、或将函数绑定到指定元素的 click 事件 |
| dblclick() | 触发、或将函数绑定到指定元素的 double click 事件 |
| delegate() | 向匹配元素的当前或未来的子元素附加一个或多个事件处理器 |
| die() | 移除所有通过 live() 函数添加的事件处理程序。 |
| error() | 触发、或将函数绑定到指定元素的 error 事件 |
| event.isDefaultPrevented() | 返回 event 对象上是否调用了 event.preventDefault()。 |
| event.pageX | 相对于文档左边缘的鼠标位置。 |
| event.pageY | 相对于文档上边缘的鼠标位置。 |
| event.preventDefault() | 阻止事件的默认动作。 |
| event.result | 包含由被指定事件触发的事件处理器返回的最后一个值。 |
| event.target | 触发该事件的 DOM 元素。 |
| event.timeStamp | 该属性返回从 1970 年 1 月 1 日到事件发生时的毫秒数。 |

| | |
|---|---|
| event.type | 描述事件的类型。 |
| event.which | 指示按了哪个键或按钮。 |
| focus() | 触发、或将函数绑定到指定元素的 focus 事件 |
| keydown() | 触发、或将函数绑定到指定元素的 key down 事件 |
| keypress() | 触发、或将函数绑定到指定元素的 key press 事件 |
| keyup() | 触发、或将函数绑定到指定元素的 key up 事件 |
| live() | 为当前或未来的匹配元素添加一个或多个事件处理器 |
| load() | 触发、或将函数绑定到指定元素的 load 事件 |
| mousedown() | 触发、或将函数绑定到指定元素的 mouse down 事件 |
| mouseenter() | 触发、或将函数绑定到指定元素的 mouse enter 事件 |
| mouseleave() | 触发、或将函数绑定到指定元素的 mouse leave 事件 |
| mousemove() | 触发、或将函数绑定到指定元素的 mouse move 事件 |
| mouseout() | 触发、或将函数绑定到指定元素的 mouse out 事件 |
| mouseover() | 触发、或将函数绑定到指定元素的 mouse over 事件 |
| mouseup() | 触发、或将函数绑定到指定元素的 mouse up 事件 |
| one() | 向匹配元素添加事件处理器。每个元素只能触发一次该处理器。 |
| ready() | 文档就绪事件（当 HTML 文档就绪可用时） |
| resize() | 触发、或将函数绑定到指定元素的 resize 事件 |
| scroll() | 触发、或将函数绑定到指定元素的 scroll 事件 |
| select() | 触发、或将函数绑定到指定元素的 select 事件 |
| submit() | 触发、或将函数绑定到指定元素的 submit 事件 |
| toggle() | 绑定两个或多个事件处理器函数，当发生轮流的 click 事件时执行。 |
| trigger() | 所有匹配元素的指定事件 |
| triggerHandler() | 第一个被匹配元素的指定事件 |
| unbind() | 从匹配元素移除一个被添加的事件处理器 |
| undelegate() | 从匹配元素移除一个被添加的事件处理器，现在或将来 |
| unload() | 触发、或将函数绑定到指定元素的 unload 事件 |

# 参阅

教程：jQuery 元素选择器语法

# jQuery 事件 - bind() 方法

## 实例

当点击鼠标时，隐藏或显示 p 元素：

```
$("button").bind("click",function(){
  $("p").slideToggle();
});
```

## 定义和用法

bind() 方法为被选元素添加一个或多个事件处理程序，并规定事件发生时运行的函数。

## 将事件和函数绑定到元素

规定向被选元素添加的一个或多个事件处理程序，以及当事件发生时运行的函数。

### 语法

```
$(_selector_).bind(_event_,_data_,_function_)
```

| 参数 | 描述 |
|---|---|
| *event* | 必需。规定添加到元素的一个或多个事件。由空格分隔多个事件。必须是有效的事件。 |
| *data* | 可选。规定传递到函数的额外数据。 |
| *function* | 必需。规定当事件发生时运行的函数。 |

### 替代语法

```
$(_selector_).bind({_event_:_function_, _event_:_function_, ...})
```

| 参数 | 描述 |
|---|---|
| {*event*:*function*, *event*:*function*, ...} | 必需。规定事件映射，其中包含一个或多个添加到元素的事件，以及当事件发生时运行的函数。 |

# jQuery 事件 - blur() 方法

## 实例

当输入域失去焦点 (blur) 时改变其颜色：

```
$("input").blur(function(){
  $("input").css("background-color","#D6D6FF");
});
```

## 定义和用法

当元素失去焦点时发生 blur 事件。

blur() 函数触发 blur 事件，或者如果设置了 *function* 参数，该函数也可规定当发生 blur 事件时执行的代码。

提示：早前，blur 事件仅发生于表单元素上。在新浏览器中，该事件可用于任何元素。

## 触发 blur 事件

触发被选元素的 blur 事件。

### 语法

```
$(_selector_).blur()
```

## 将函数绑定到 blur 事件

规定当被选元素的 blur 事件发生时运行的函数。

### 语法

```
$(_selector_).blur(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当 blur 事件发生时运行的函数。 |

# jQuery 事件 - change() 方法

## 实例

当输入域发生变化时改变其颜色：

```
$(".field").change(function(){
  $(this).css("background-color","#FFFFCC");
});
```

## 定义和用法

当元素的值发生改变时，会发生 change 事件。

该事件仅适用于文本域（text field），以及 textarea 和 select 元素。

change() 函数触发 change 事件，或规定当发生 change 事件时运行的函数。

注释：当用于 select 元素时，change 事件会在选择某个选项时发生。当用于 text field 或 text area 时，该事件会在元素失去焦点时发生。

## 触发 change 事件

触发被选元素的 change 事件。

### 语法

```
$(_selector_).change()
```

## 将函数绑定到 change 事件

规定当被选元素的 change 事件发生时运行的函数。

### 语法

```
$(_selector_).change(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当 change 事件发生时运行的函数。 |

# jQuery 事件 - click() 方法

## 实例

当点击按钮时，隐藏或显示元素：

```
$("button").click(function(){
  $("p").slideToggle();
});
```

## 定义和用法

当点击元素时，会发生 click 事件。

当鼠标指针停留在元素上方，然后按下并松开鼠标左键时，就会发生一次 click。

click() 方法触发 click 事件，或规定当发生 click 事件时运行的函数。

## 触发 click 事件

### 语法

```
$(_selector_).click()
```

## 将函数绑定到 click 事件

### 语法

```
$(_selector_).click(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当发生 click 事件时运行的函数。 |

# jQuery 事件 - dblclick() 方法

## 实例

当双击按钮时，隐藏或显示元素：

```
$("button").dblclick(function(){
  $("p").slideToggle();
});
```

## 定义和用法

当双击元素时，会发生 dblclick 事件。

当鼠标指针停留在元素上方，然后按下并松开鼠标左键时，就会发生一次 click。

在很短的时间内发生两次 click，即是一次 double click 事件。

dblclick() 方法触发 dblclick 事件，或规定当发生 dblclick 事件时运行的函数。

提示：如果把 dblclick 和 click 事件应用于同一元素，可能会产生问题。

## 触发 **dblclick** 事件

### 语法

```
$(_selector_).dblclick()
```

## 将函数绑定到 **dblclick** 事件

### 语法

```
$(_selector_).dblclick(_function_)
```

| 参数 | 描述 |
|---|---|
| *function* | 可选。规定当发生 dblclick 事件时运行的函数。 |

# jQuery 事件 - delegate() 方法

## 实例

当点击鼠标时，隐藏或显示 p 元素：

```
$("div").delegate("button","click",function(){
  $("p").slideToggle();
});
```

## 定义和用法

delegate() 方法为指定的元素（属于被选元素的子元素）添加一个或多个事件处理程序，并规定当这些事件发生时运行的函数。

使用 delegate() 方法的事件处理程序适用于当前或未来的元素（比如由脚本创建的新元素）。

### 语法

```
$(_selector_).delegate(_childSelector_,_event_,_data_,_function_)
```

| 参数 | 描述 |
|---|---|
| *childSelector* | 必需。规定要附加事件处理程序的一个或多个子元素。 |
| *event* | 必需。规定附加到元素的一个或多个事件。由空格分隔多个事件值。必须是有效的事件。 |
| *data* | 可选。规定传递到函数的额外数据。 |
| *function* | 必需。规定当事件发生时运行的函数。 |

## 亲自试一试 - 实例

向未来的元素添加事件处理程序

如何使用 delegate() 方法向尚未创建的元素添加事件处理程序。

# jQuery 事件 - die() 方法

## 实例

移除所有通过 live() 方法向 p 元素添加的事件处理程序：

```
$("p").die();
```

## 定义和用法

die() 方法移除所有通过 live() 方法向指定元素添加的一个或多个事件处理程序。

### 语法

```
$(_selector_).die(_event_,_function_)
```

| 参数 | 描述 |
| --- | --- |
| *event* | 必需。规定要移除的一个或多个事件处理程序。由空格分隔多个事件值。必须是有效的事件。 |
| *function* | 可选。规定要移除的特定函数。 |

## 亲自试一试 - 实例

移除通过 live() 添加的特定函数

如何使用 live() 方法移除事件处理程序的特定函数。

# jQuery 事件 - error() 方法

## 实例

如果图像不存在，则用一段预定义的文本取代它：

```
$("img").error(function(){
  $("img").replaceWith("

Missing image!

");
});
```

## 定义和用法

当元素遇到错误（没有正确载入）时，发生 error 事件。

error() 方法触发 error 事件，或规定当发生 error 事件时运行的函数。

提示：该方法是 bind('error', handler) 的简写方式。

## 触发 error 事件

### 语法

```
$(_selector_).error()
```

## 将函数绑定到 error 事件

### 语法

```
$(_selector_).error(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当发生 error 事件时运行的函数。 |

# jQuery 事件 - isDefaultPrevented() 方法

## 实例

防止链接打开 URL，并声明来自 isDefaultPrevented() 的结果：

```
$("a").click(function(event){
   event.preventDefault();
  alert("Default prevented: " + event.isDefaultPrevented());
});
```

## 定义和用法

isDefaultPrevented() 方法返回指定的 event 对象上是否调用了 preventDefault() 方法。

### 语法

```
_event_.isDefaultPrevented()
```

| 参数 | 描述 |
| --- | --- |
| *event* | 必需。规定需要检查的事件。这个 *event* 参数来自事件绑定函数。 |

# jQuery 事件 - pageX 属性

## 实例

显示鼠标指针的位置：

```
$(document).mousemove(function(e){
  $("span").text("X: " + e.pageX + ", Y: " + e.pageY);
});
```

## 定义和用法

pageX() 属性是鼠标指针的位置，相对于文档的左边缘。

### 语法

```
_event_.pageX
```

| 参数 | 描述 |
|---|---|
| *event* | 必需。规定要使用的事件。这个 *event* 参数来自事件绑定函数。 |

# jQuery 事件 - pageY 属性

## 实例

显示鼠标指针的位置：

```
$(document).mousemove(function(e){
  $("span").text("X: " + e.pageX + ", Y: " + e.pageY);
});
```

## 定义和用法

pageY() 属性是鼠标指针的位置，相对于文档的上边缘。

### 语法

```
_event_.pageY
```

| 参数 | 描述 |
| --- | --- |
| *event* | 必需。规定要使用的事件。这个 *event* 参数来自事件绑定函数。 |

# jQuery 事件 - preventDefault() 方法

## 实例

防止链接打开 URL：

```
$("a").click(function(event){
  event.preventDefault();
});
```

## 定义和用法

preventDefault() 方法阻止元素发生默认的行为（例如，当点击提交按钮时阻止对表单的提交）。

### 语法

```
_event_.preventDefault()
```

| 参数 | 描述 |
| --- | --- |
| *event* | 必需。规定阻止哪个事件的默认动作。这个 *event* 参数来自事件绑定函数。 |

# jQuery 事件 - result 属性

## 实例

显示最后一次点击事件返回的结果：

```
$("button").click(function(e) {
  $("p").html(e.result);
});
```

## 定义和用法

result 属性包含由被指定事件触发的事件处理器返回的最后一个值，除非这个值未定义。

### 语法

```
_event_.result
```

| 参数 | 描述 |
|---|---|
| *event* | 必需。规定返回的最后一个值来自哪个事件。这个 *event* 参数来自事件绑定函数。 |

# jQuery 事件 - target 属性

## 实例

显示哪个 DOM 元素触发了事件：

```
$("p, button, h1, h2").click(function(event){
  $("div").html("Triggered by a " + event.target.nodeName + " element.");
});
```

## 定义和用法

target 属性规定哪个 DOM 元素触发了该事件。

### 语法

```
_event_.target
```

| 参数 | 描述 |
| --- | --- |
| *event* | 必需。规定需要检查的事件。这个 *event* 参数来自事件绑定函数。 |

# jQuery 事件 - timeStamp 属性

## 实例

显示出当对按钮元素的点击事件发生时的时间戳：

```
$("button").click(function(event){
  $("span")html(event.timeStamp);
});
```

## 定义和用法

timeStamp 属性包含从 1970 年 1 月 1 日到事件被触发时的毫秒数。

### 语法

```
_event_.timeStamp
```

| 参数 | 描述 |
| --- | --- |
| *event* | 必需。规定返回该时间戳的事件。这个 *event* 参数来自事件绑定函数。 |

# jQuery 事件 - type 属性

## 实例

显示触发了哪种类型的事件：

```
$("p").bind('click dblclick mouseover mouseout',function(event){
  $("div").html("Event: " + event.type);
});
```

## 定义和用法

type 属性描述触发哪种事件类型。

### 语法

```
_event_.type
```

| 参数 | 描述 |
| --- | --- |
| *event* | 必需。规定要检查的事件。这个 *event* 参数来自事件绑定函数。 |

# jQuery 事件 - which 属性

## 实例

显示按了哪个键：

```
$("input").keydown(function(event){
  $("div").html("Key: " + event.which);
});
```

## 定义和用法

which 属性指示按了哪个键或按钮。

### 语法

```
_event_.which
```

| 参数 | 描述 |
| --- | --- |
| *event* | 必需。规定要检查的事件。这个 *event* 参数来自事件绑定函数。 |

# jQuery 事件 - focus() 方法

## 实例

当输入框获得焦点时，改变它的背景色：

```
$("input").focus(function(){
  $("input").css("background-color","#FFFFCC");
});
```

## 定义和用法

当元素获得焦点时，发生 focus 事件。

当通过鼠标点击选中元素或通过 tab 键定位到元素时，该元素就会获得焦点。

focus() 方法触发 focus 事件，或规定当发生 focus 事件时运行的函数。

## 触发 focus 事件

### 语法

```
$(_selector_).focus()
```

## 将函数绑定到 focus 事件

### 语法

```
$(_selector_).focus(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当发生 focus 事件时运行的函数。 |

# jQuery 事件 - keydown() 方法

## 实例

当按下按键时，改变文本域的颜色：

```
$("input").keydown(function(){
  $("input").css("background-color","#FFFFCC");
});
```

## 定义和用法

完整的 key press 过程分为两个部分：1. 按键被按下；2. 按键被松开。

当按钮被按下时，发生 keydown 事件。

keydown() 方法触发 keydown 事件，或规定当发生 keydown 事件时运行的函数。

注释：如果在文档元素上进行设置，则无论元素是否获得焦点，该事件都会发生。

提示：请使用 .which 属性来确定按下了哪个按键（亲自试一试）。

## 触发 keydown 事件

### 语法

```
$(_selector_).keydown()
```

## 将函数绑定到 keydown 事件

### 语法

```
$(_selector_).keydown(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当发生 keydown 事件时运行的函数。 |

# jQuery 事件 - keypress() 方法

## 实例

计算在输入域中的按键次数：

```
$("input").keydown(function(){
  $("span").text(i+=1);
});
```

## 定义和用法

keypress 事件与 keydown 事件类似。当按钮被按下时，会发生该事件。它发生在当前获得焦点的元素上。

不过，与 keydown 事件不同，每插入一个字符，就会发生 keypress 事件。

keypress() 方法触发 keypress 事件，或规定当发生 keypress 事件时运行的函数。

注释：如果在文档元素上进行设置，则无论元素是否获得焦点，该事件都会发生。

## 触发 keypress 事件

### 语法

```
$(_selector_).keypress()
```

## 将函数绑定到 keypress 事件

### 语法

```
$(_selector_).keypress(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当发生 keypress 事件时运行的函数。 |

# jQuery 事件 - keyup() 方法

## 实例

当按下按键时，改变文本域的颜色：

```
$("input").keyup(function(){
  $("input").css("background-color","#D6D6FF");
});
```

## 定义和用法

完整的 key press 过程分为两个部分，按键被按下，然后按键被松开并复位。

当按钮被松开时，发生 keyup 事件。它发生在当前获得焦点的元素上。

keyup() 方法触发 keyup 事件，或规定当发生 keyup 事件时运行的函数。

注释：如果在文档元素上进行设置，则无论元素是否获得焦点，该事件都会发生。

提示：请使用 .which 属性来确定按下了哪个按键（亲自试一试）。

## 触发 keyup 事件

### 语法

```
$(_selector_).keyup()
```

## 将函数绑定到 keyup 事件

### 语法

```
$(_selector_).keyup(_function_)
```

| 参数 | 描述 |
|---|---|
| *function* | 可选。规定当发生 keyup 事件时运行的函数。 |

# jQuery 事件 - live() 方法

## 实例

当点击按钮时，隐藏或显示 p 元素：

```
$("button").live("click",function(){
  $("p").slideToggle();
});
```

## 定义和用法

live() 方法为被选元素附加一个或多个事件处理程序，并规定当这些事件发生时运行的函数。

通过 live() 方法附加的事件处理程序适用于匹配选择器的当前及未来的元素（比如由脚本创建的新元素）。

### 语法

```
$(_selector_).live(_event_,_data_,_function_)
```

| 参数 | 描述 |
| --- | --- |
| *event* | 必需。规定附加到元素的一个或多个事件。由空格分隔多个事件。必须是有效的事件。 |
| *data* | 可选。规定传递到该函数的额外数据。 |
| *function* | 必需。规定当事件发生时运行的函数。 |

## 亲自试一试 - 实例

向未来的元素添加事件处理器

如何使用 live() 方法向尚未创建的元素添加事件处理器。

# jQuery 事件 - load() 方法

## 实例

当图像加载时，改变 div 元素的文本：

```
$("img").load(function(){
  $("div").text("Image loaded");
});
```

## 定义和用法

当指定的元素（及子元素）已加载时，会发生 load() 事件。

该事件适用于任何带有 URL 的元素（比如图像、脚本、框架、内联框架）。

根据不同的浏览器（Firefox 和 IE），如果图像已被缓存，则也许不会触发 load 事件。

注释：还存在一个名为 load() 的 jQuery Ajax 方法，根据不同的参数而定。

## 语法

```
$(_selector_).load(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 必需。规定当指定元素加载完成时运行的函数。 |

# jQuery 事件 - mousedown() 方法

## 实例

当按下鼠标按钮时，隐藏或显示元素：

```
$("button").mousedown(function(){
  $("p").slideToggle();
});
```

## 定义和用法

当鼠标指针移动到元素上方，并按下鼠标按键时，会发生 mousedown 事件。

与 click 事件不同，mousedown 事件仅需要按键被按下，而不需要松开即可发生。

mousedown() 方法触发 mousedown 事件，或规定当发生 mousedown 事件时运行的函数。

## 触发 mousedown 事件

### 语法

```
$(_selector_).mousedown()
```

## 将函数绑定到 mousedown 事件

### 语法

```
$(_selector_).mousedown(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当发生 mousedown 事件时运行的函数。 |

# jQuery 事件 - mouseenter() 方法

## 实例

当鼠标指针进入（穿过）元素时，改变元素的背景色：

```
$("p").mouseenter(function(){
  $("p").css("background-color","yellow");
});
```

## 定义和用法

当鼠标指针穿过元素时，会发生 mouseenter 事件。

该事件大多数时候会与 mouseleave 事件一起使用。

mouseenter() 方法触发 mouseenter 事件，或规定当发生 mouseenter 事件时运行的函数。

注释：与 mouseover 事件不同，只有在鼠标指针穿过被选元素时，才会触发 mouseenter 事件。如果鼠标指针穿过任何子元素，同样会触发 mouseover 事件。请看下面例子的演示。

亲自试一试：mouseenter 与 mouseover 的不同

## 触发 mouseenter 事件

### 语法

```
$(_selector_).mouseenter()
```

## 将函数绑定到 mouseenter 事件

### 语法

```
$(_selector_).mouseenter(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当发生 mouseenter 事件时运行的函数。 |

# jQuery 事件 - mouseleave() 方法

## 实例

当鼠标指针离开元素时，改变元素的背景色：

```
$("p").mouseleave(function(){
  $("p").css("background-color","#E9E9E4");
});
```

## 定义和用法

当鼠标指针离开元素时，会发生 mouseleave 事件。

该事件大多数时候会与 mouseenter 事件一起使用。

mouseleave() 方法触发 mouseleave 事件，或规定当发生 mouseleave 事件时运行的函数。

注释：与 mouseout 事件不同，只有在鼠标指针离开被选元素时，才会触发 mouseleave 事件。如果鼠标指针离开任何子元素，同样会触发 mouseout 事件。请看下面例子的演示。

亲自试一试：mouseleave 与 mouseout 的不同

## 触发 mouseleave 事件

### 语法

```
$(_selector_).mouseleave()
```

## 将函数绑定到 mouseleave 事件

### 语法

```
$(_selector_).mouseleave(_function_)
```

| 参数 | 描述 |
|---|---|
| *function* | 可选。规定当发生 mouseleave 事件时运行的函数。 |

# jQuery 事件 - mousemove() 方法

## 实例

获得鼠标指针在页面中的位置：

```
$(document).mousemove(function(e){
  $("span").text(e.pageX + ", " + e.pageY);
});
```

## 定义和用法

当鼠标指针在指定的元素中移动时，就会发生 mousemove 事件。

mousemove() 方法触发 mousemove 事件，或规定当发生 mousemove 事件时运行的函数。

注意：用户把鼠标移动一个像素，就会发生一次 mousemove 事件。处理所有 mousemove 事件会耗费系统资源。请谨慎使用该事件。

## 触发 mousemove 事件

### 语法

```
$(_selector_).mousemove()
```

## 将函数绑定到 mousemove 事件

### 语法

```
$(_selector_).mousemove(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当发生 mousemove 事件时运行的函数。 |

# jQuery 事件 - mouseout() 方法

## 实例

当鼠标从元素上移开时，改变元素的背景色：

```
$("p").mouseout(function(){
  $("p").css("background-color","#E9E9E4");
});
```

## 定义和用法

当鼠标指针从元素上移开时，发生 mouseout 事件。

该事件大多数时候会与 mouseover 事件一起使用。

mouseout() 方法触发 mouseout 事件，或规定当发生 mouseout 事件时运行的函数。

注释：与 mouseleave 事件不同，不论鼠标指针离开被选元素还是任何子元素，都会触发 mouseout 事件。只有在鼠标指针离开被选元素时，才会触发 mouseleave 事件。请看下面例子的演示。

亲自试一试：mouseleave 与 mouseout 的不同

## 触发 mouseout 事件

### 语法

```
$(_selector_).mouseout()
```

## 将函数绑定到 mouseout 事件

### 语法

```
$(_selector_).mouseout(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当发生 mouseout 事件时运行的函数。 |

# jQuery 事件 - mouseover() 方法

## 实例

当鼠标指针位于元素上方时时，改变元素的背景色：

```
$("p").mouseover(function(){
  $("p").css("background-color","yellow");
});
```

## 定义和用法

当鼠标指针位于元素上方时，会发生 mouseover 事件。

该事件大多数时候会与 mouseout 事件一起使用。

mouseover() 方法触发 mouseover 事件，或规定当发生 mouseover 事件时运行的函数。

注释：与 mouseenter 事件不同，不论鼠标指针穿过被选元素或其子元素，都会触发 mouseover 事件。只有在鼠标指针穿过被选元素时，才会触发 mouseenter 事件。请看下面例子的演示。

亲自试一试：mouseenter 与 mouseover 的不同

## 触发 mouseover 事件

### 语法

```
$(_selector_).mouseover()
```

## 将函数绑定到 mouseover 事件

### 语法

```
$(_selector_).mouseover(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当发生 mouseover 事件时运行的函数。 |

# jQuery 事件 - mouseup() 方法

## 实例

当松开鼠标按钮时，隐藏或显示元素：

```
$("button").mouseup(function(){
  $("p").slideToggle();
});
```

## 定义和用法

当在元素上放松鼠标按钮时，会发生 mouseup 事件。

与 click 事件不同，mouseup 事件仅需要放松按钮。当鼠标指针位于元素上方时，放松鼠标按钮就会触发该事件。

mouseup() 方法触发 mouseup 事件，或规定当发生 mouseup 事件时运行的函数。

## 触发 mouseup 事件

### 语法

```
$(_selector_).mouseup()
```

## 将函数绑定到 mouseup 事件

### 语法

```
$(_selector_).mouseup(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当发生 mouseup 事件时运行的函数。 |

# jQuery 事件 - one() 方法

## 实例

当点击 p 元素时，增加该元素的文本大小：

```
$("p").one("click",function(){
  $(this).animate({fontSize:"+=6px"});
});
```

## 定义和用法

one() 方法为被选元素附加一个或多个事件处理程序，并规定当事件发生时运行的函数。

当使用 one() 方法时，每个元素只能运行一次事件处理器函数。

### 语法

```
$(_selector_).one(_event_,_data_,_function_)
```

| 参数 | 描述 |
| --- | --- |
| *event* | 必需。规定添加到元素的一个或多个事件。由空格分隔多个事件。必须是有效的事件。 |
| *data* | 可选。规定传递到函数的额外数据。 |
| *function* | 必需。规定当事件发生时运行的函数。 |

# jQuery 事件 - ready() 方法

## 实例

在文档加载后激活函数：

```
$(document).ready(function(){
  $(".btn1").click(function(){
    $("p").slideToggle();
  });
});
```

## 定义和用法

当 DOM（文档对象模型） 已经加载，并且页面（包括图像）已经完全呈现时，会发生 ready 事件。

由于该事件在文档就绪后发生，因此把所有其他的 jQuery 事件和函数置于该事件中是非常好的做法。正如上面的例子中那样。

ready() 函数规定当 ready 事件发生时执行的代码。

ready() 函数仅能用于当前文档，因此无需选择器。

允许使用以下三种语法：

### 语法 1

```
$(document).ready(_function_)
```

### 语法 2

```
$().ready(_function_)
```

### 语法 3

```
$(_function_)
```

| 参数 | 描述 |
| --- | --- |
| function | 必需。规定当文档加载后要运行的函数。 |

# 提示和注释

提示：ready() 函数不应与 <body onload=""> 一起使用。

# jQuery 事件 - resize() 方法

## 实例

对浏览器窗口调整大小进行计数：

```
$(window).resize(function() {
  $('span').text(x+=1);
});
```

## 定义和用法

当调整浏览器窗口的大小时，发生 resize 事件。

resize() 方法触发 resize 事件，或规定当发生 resize 事件时运行的函数。

## 触发 resize 事件

### 语法

```
$(_selector_).resize()
```

## 将函数绑定到 resize 事件

### 语法

```
$(_selector_).resize(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当发生 resize 事件时运行的函数。 |

# jQuery 事件 - scroll() 方法

## 实例

对元素滚动的次数进行计数：

```
$("div").scroll(function() {
  $("span").text(x+=1);
});
```

## 定义和用法

当用户滚动指定的元素时，会发生 scroll 事件。

scroll 事件适用于所有可滚动的元素和 window 对象（浏览器窗口）。

scroll() 方法触发 scroll 事件，或规定当发生 scroll 事件时运行的函数。

## 触发 scroll 事件

### 语法

```
$(_selector_).scroll()
```

## 将函数绑定到 scroll 事件

### 语法

```
$(_selector_).scroll(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当发生 scroll 事件时运行的函数。 |

# jQuery 事件 - select() 方法

## 实例

在文本域后添加文本，以显示出提示文本：

```
$("input").select(function(){
  $("input").after(" Text marked!");
});
```

## 定义和用法

当 textarea 或文本类型的 input 元素中的文本被选择时，会发生 select 事件。

select() 方法触发 select 事件，或规定当发生 select 事件时运行的函数。

## 触发 select 事件

### 语法

```
$(_selector_).select()
```

## 将函数绑定到 select 事件

### 语法

```
$(_selector_).select(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当发生 select 事件时运行的函数。 |

# jQuery 事件 - submit() 方法

## 实例

当提交表单时，显示警告框：

```
$("form").submit(function(e){
  alert("Submitted");
});
```

## 定义和用法

当提交表单时，会发生 submit 事件。

该事件只适用于表单元素。

submit() 方法触发 submit 事件，或规定当发生 submit 事件时运行的函数。

## 触发 submit 事件

### 语法

```
$(_selector_).submit()
```

## 将函数绑定到 submit 事件

### 语法

```
$(_selector_).submit(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 可选。规定当发生 submit 事件时运行的函数。 |

## 亲自试一试 - 实例

阻止提交按钮的默认 action

使用 preventDefault() 函数来阻止对表单的提交。

# jQuery 事件 - toggle() 方法

## 实例

切换不同的背景色：

```
$("p").toggle(
  function(){
  $("body").css("background-color","green");},
  function(){
  $("body").css("background-color","red");},
  function(){
  $("body").css("background-color","yellow");}
);
```

## 定义和用法

toggle() 方法用于绑定两个或多个事件处理器函数，以响应被选元素的轮流的 click 事件。

该方法也可用于切换被选元素的 hide() 与 show() 方法。

## 向 Toggle 事件绑定两个或更多函数

当指定元素被点击时，在两个或多个函数之间轮流切换。

如果规定了两个以上的函数，则 toggle() 方法将切换所有函数。例如，如果存在三个函数，则第一次点击将调用第一个函数，第二次点击调用第二个函数，第三次点击调用第三个函数。第四次点击再次调用第一个函数，以此类推。

### 语法

```
$(_selector_).toggle(_function1()_,_function2()_,_functionN()_,...)
```

| 参数 | 描述 |
| --- | --- |
| *function1()* | 必需。规定当元素在每偶数次被点击时要运行的函数。 |
| *function2()* | 必需。规定当元素在每奇数次被点击时要运行的函数。 |
| *functionN(),...* | 可选。规定需要切换的其他函数。 |

## 切换 Hide() 和 Show()

检查每个元素是否可见。

如果元素已隐藏，则运行 show()。如果元素可见，则元素 hide()。这样就可以创造切换效果。

## 语法

```
$(_selector_).toggle(_speed_,_callback_)
```

| 参数 | 描述 |
| --- | --- |
| speed | 可选。规定 hide/show 效果的速度。默认是 "0"。可能的值： 毫秒（比如 1500） "slow" "normal" "fast" |
| callback | 可选。当 toggle() 方法完成时执行的函数。如需学习更多有关 callback 的知识，请访问我们的 Callback 函数教程。 |

# 显示或隐藏元素

规定是否只显示或只隐藏所有匹配的元素。

## 语法

```
$(_selector_).toggle(_switch_)
```

| 参数 | 描述 |
| --- | --- |
| *switch* | 必需。布尔值，规定 toggle() 是否应只显示或只隐藏所有被选元素。 `true` - 显示元素 `false` - 隐藏元素 |

# jQuery 事件 - trigger() 方法

## 实例

触发 input 元素的 select 事件：

```
$("button").click(function(){
  $("input").trigger("select");
});
```

## 定义和用法

trigger() 方法触发被选元素的指定事件类型。

## 触发事件

规定被选元素要触发的事件。

### 语法

```
$(_selector_).trigger(_event_,[_param1_,_param2_,...])
```

| 参数 | 描述 |
|---|---|
| *event* | 必需。规定指定元素要触发的事件。可以使自定义事件（使用 bind() 函数来附加），或者任何标准事件。 |
| [*param1*,*param2*,...] | 可选。传递到事件处理程序的额外参数。额外的参数对自定义事件特别有用。 |

## 使用 **Event** 对象来触发事件

规定使用事件对象的被选元素要触发的事件。

### 语法

```
$(_selector_).trigger(_eventObj_)
```

| 参数 | 描述 |
| --- | --- |
| *eventObj* | 必需。规定事件发生时运行的函数。 |

# jQuery 事件 - triggerHandler() 方法

## 实例

触发 input 元素的 select 事件：

```
$("button").click(function(){
  $("input").triggerHandler("select");
});
```

## 定义和用法

triggerHandler() 方法触发被选元素的指定事件类型。但不会执行浏览器默认动作，也不会产生事件冒泡。

triggerHandler() 方法与 trigger() 方法类似。不同的是它不会触发事件（比如表单提交）的默认行为，而且只影响第一个匹配元素。

### 与 **trigger()** 方法相比的不同之处

- 它不会引起事件（比如表单提交）的默认行为
- .trigger() 会操作 jQuery 对象匹配的所有元素，而 .triggerHandler() 只影响第一个匹配元素。
- 由 .triggerHandler() 创建的事件不会在 DOM 树中冒泡；如果目标元素不直接处理它们，则不会发生任何事情。
- 该方法的返回的是事件处理函数的返回值，而不是具有可链性的 jQuery 对象。此外，如果没有处理程序被触发，则这个方法返回 undefined。

## 触发事件

规定被选元素要触发的事件。

### 语法

```
$(_selector_).triggerHandler(_event_,[_param1_,_param2_,...])
```

| 参数 | 描述 |
|---|---|
| *event* | 必需。规定指定元素要触发的事件。 |
| [*param1*,*param2*,...] | 可选。传递到事件处理程序的额外参数。 |

# jQuery 事件 - unbind() 方法

## 实例

移除所有 p 元素的事件处理器：

```
$("button").click(function(){
  $("p").unbind();
});
```

## 定义和用法

unbind() 方法移除被选元素的事件处理程序。

该方法能够移除所有的或被选的事件处理程序，或者当事件发生时终止指定函数的运行。

ubind() 适用于任何通过 jQuery 附加的事件处理程序。

## 取消绑定元素的事件处理程序和函数

规定从指定元素上删除的一个或多个事件处理程序。

如果没有规定参数，unbind() 方法会删除指定元素的所有事件处理程序。

### 语法

```
$(_selector_).unbind(_event_,_function_)
```

| 参数 | 描述 |
| --- | --- |
| *event* | 可选。规定删除元素的一个或多个事件由空格分隔多个事件值。如果只规定了该参数，则会删除绑定到指定事件的所有函数。 |
| *function* | 可选。规定从元素的指定事件取消绑定的函数名。 |

## 使用 Event 对象来取消绑定事件处理程序

规定要删除的事件对象。用于对自身内部的事件取消绑定（比如当事件已被触发一定次数之后，删除事件处理程序）。

如果未规定参数，则 unbind() 方法会删除指定元素的所有事件处理程序。

## 语法

```
$(_selector_).unbind(_eventObj_)
```

| 参数 | 描述 |
| --- | --- |
| *eventObj* | 可选。规定要使用的事件对象。这个 eventObj 参数来自事件绑定函数。 |

# 亲自试一试 - 实例

取消绑定特定的函数

如何使用 unbind() 方法取消绑定元素指定事件的具体函数。

# jQuery 事件 - undelegate() 方法

## 实例

从所有元素删除由 delegate() 方法添加的所有事件处理器：

```
$("body").undelegate();
```

## 定义和用法

undelegate() 方法删除由 delegate() 方法添加的一个或多个事件处理程序。

### 语法

```
$(_selector_).undelegate(_selector_,_event_,_function_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 可选。规定需要删除事件处理程序的选择器。 |
| *event* | 可选。规定需要删除处理函数的一个或多个事件类型。 |
| *function* | 可选。规定要删除的具体事件处理函数。 |

## 亲自试一试 - 实例

删除事件处理程序，由 delegate() 添加，来自具体选择器

如何使用 undelegate() 方法从指定元素删除所有事件处理程序。

删除指定事件类型的事件处理程序，由 delegate() 添加，来自指定元素

如何使用 undelegate() 方法从指定元素删除具体事件类型的所有事件处理程序。

删除由 delegate() 添加的具体函数

如何使用 undelegate() 方法为事件处理程序删除来自特定事件类型的特定函数。

# jQuery 事件 - unload 属性

## 实例

当用户点击链接离开本页时，弹出一个消息框：

```
$(window).unload(function(){
  alert("Goodbye!");
});
```

## 定义和用法

当用户离开页面时，会发生 unload 事件。

具体来说，当发生以下情况时，会发出 unload 事件：

- 点击某个离开页面的链接
- 在地址栏中键入了新的 URL
- 使用前进或后退按钮
- 关闭浏览器
- 重新加载页面

unload() 方法将事件处理程序绑定到 unload 事件。

unload() 方法只应用于 window 对象。

## 语法

```
_event_.unload(_function_)
```

| 参数 | 描述 |
| --- | --- |
| *function* | 必需。规定当触发 unload 事件时运行的函数。 |

# jQuery 参考手册 - 效果

## jQuery 效果函数

| 方法 | 描述 |
| --- | --- |
| animate() | 对被选元素应用"自定义"的动画 |
| clearQueue() | 对被选元素移除所有排队的函数（仍未运行的） |
| delay() | 对被选元素的所有排队函数（仍未运行）设置延迟 |
| dequeue() | 运行被选元素的下一个排队函数 |
| fadeIn() | 逐渐改变被选元素的不透明度，从隐藏到可见 |
| fadeOut() | 逐渐改变被选元素的不透明度，从可见到隐藏 |
| fadeTo() | 把被选元素逐渐改变至给定的不透明度 |
| hide() | 隐藏被选的元素 |
| queue() | 显示被选元素的排队函数 |
| show() | 显示被选的元素 |
| slideDown() | 通过调整高度来滑动显示被选元素 |
| slideToggle() | 对被选元素进行滑动隐藏和滑动显示的切换 |
| slideUp() | 通过调整高度来滑动隐藏被选元素 |
| stop() | 停止在被选元素上运行动画 |
| toggle() | 对被选元素进行隐藏和显示的切换 |

# jQuery 效果 - animate() 方法

## 实例

改变 "div" 元素的高度：

```
$(".btn1").click(function(){
  $("#box").animate({height:"300px"});
});
```

## 定义和用法

animate() 方法执行 CSS 属性集的自定义动画。

该方法通过CSS样式将元素从一个状态改变为另一个状态。CSS属性值是逐渐改变的，这样就可以创建动画效果。

只有数字值可创建动画（比如 "margin:30px"）。字符串值无法创建动画（比如 "background-color:red"）。

注释：使用 "+=" 或 "-=" 来创建相对动画（relative animations）。

### 语法 1

```
$(_selector_).animate(_styles_,_speed_,_easing_,_callback_)
```

| 参数 | 描述 |
| --- | --- |
| styles | 必需。规定产生动画效果的 CSS 样式和值。可能的 CSS 样式值（提供实例）： `backgroundPosition` `borderWidth` `borderBottomWidth` `borderLeftWidth` `borderRightWidth` `borderTopWidth` `borderSpacing` `margin` `marginBottom` `marginLeft` `marginRight` `marginTop` `outlineWidth` `padding` `paddingBottom` `paddingLeft` `paddingRight` `paddingTop` `height` `width` `maxHeight` `maxWidth` `minHeight` `minWidth` `font` `fontSize` `bottom` `left` `right` `top` `letterSpacing` `wordSpacing` `lineHeight` `textIndent` 注释：CSS 样式使用 DOM 名称（比如 "fontSize"）来设置，而非 CSS 名称（比如 "font-size"）。 |
| speed | 可选。规定动画的速度。默认是 "normal"。可能的值：毫秒 （比如 1500） "slow" "normal" "fast" |
| easing | 可选。规定在不同的动画点中设置动画速度的 easing 函数。内置的 easing 函数： `swing` `linear` 扩展插件中提供更多 easing 函数。 |
| callback | 可选。animate 函数执行完之后，要执行的函数。如需学习更多有关 callback 的内容，请访问我们的 jQuery Callback 这一章。 |

## 语法 2

```
$(_selector_).animate(_styles_,_options_)
```

| 参数 | 描述 |
| --- | --- |
| *styles* | 必需。规定产生动画效果的 CSS 样式和值（同上）。 |
| *options* | 可选。规定动画的额外选项。可能的值： `speed` - 设置动画的速度 `easing` - 规定要使用的 easing 函数 `callback` - 规定动画完成之后要执行的函数 `step` - 规定动画的每一步完成之后要执行的函数 `queue` - 布尔值。指示是否在效果队列中放置动画。如果为 false，则动画将立即开始 `specialEasing` - 来自 *styles* 参数的一个或多个 CSS 属性的映射，以及它们的对应 easing 函数 |

# jQuery 效果 - clearQueue() 方法

## 实例

停止当前正在运行的动画：

```
$("#stop").click(function(){
  $("#box").clearQueue();
});
```

## 定义和用法

clearQueue() 方法停止队列中所有仍未执行的函数。

与 stop() 方法不同，（只适用于动画），clearQueue() 能够清除任何排队的函数（通过 .queue() 方法添加到通用 jQuery 队列的任何函数）。

### 语法

```
$(_selector_).clearQueue(_queueName_)
```

| 参数 | 描述 |
| --- | --- |
| *queueName* | 可选。规定要停止的队列的名称。默认是 "fx"，标准效果队列。 |

# jQuery 效果 - fadeIn() 方法

## 实例

使用淡入效果来显示一个隐藏的 <p> 元素：

```
$(".btn2").click(function(){
  $("p").fadeIn();
});
```

## 定义和用法

fadeIn() 方法使用淡入效果来显示被选元素，假如该元素是隐藏的。

### 语法

```
$(_selector_).fadeIn(_speed_,_callback_)
```

| 参数 | 描述 |
|---|---|
| *speed* | 可选。规定元素从隐藏到可见的速度。默认为 "normal"。可能的值： 毫秒（比如 1500） "slow" "normal" "fast"在设置速度的情况下，元素从隐藏到可见的过程中，会逐渐地改变其透明度（这样会创造淡入效果）。 |
| *callback* | 可选。fadeIn 函数执行完之后，要执行的函数。如需学习更多有关 callback 的内容，请访问我们的 jQuery Callback 这一章。除非设置了 speed 参数，否则不能设置该参数。 |

## 提示和注释

提示：如果元素已经显示，则该效果不产生任何变化，除非规定了 callback 函数。

注释：该效果适用于通过 jQuery 隐藏的元素，或在 CSS 中声明 display:none 的元素（但不适用于 visibility:hidden 的元素）。

## 亲自试一试 - 实例

使用 speed 参数

使用 speed 参数来对淡入或淡出元素。

# jQuery 效果 - fadeOut() 方法

## 实例

使用淡出效果来隐藏一个 <p> 元素：

```
$(".btn1").click(function(){
  $("p").fadeOut();
});
```

## 定义和用法

fadeOut() 方法使用淡出效果来隐藏被选元素，假如该元素是隐藏的。

### 语法

```
$(_selector_).fadeOut(_speed_,_callback_)
```

| 参数 | 描述 |
| --- | --- |
| *speed* | 可选。规定元素从可见到隐藏的速度。默认为 "normal"。可能的值： 毫秒（比如 1500） "slow" "normal" "fast"在设置速度的情况下，元素从可见到隐藏的过程中，会逐渐地改变其透明度（这样会创造淡出效果）。 |
| *callback* | 可选。fadeOut 函数执行完之后，要执行的函数。如需学习更多有关 callback 的内容，请访问我们的 jQuery Callback 这一章。除非设置了 speed 参数，否则不能设置该参数。 |

## 提示和注释

提示：如果元素已经隐藏，则该效果不产生任何变化，除非规定了 callback 函数。

## 亲自试一试 - 实例

使用 speed 参数

使用 speed 参数来对淡入或淡出元素。

# jQuery 效果 - fadeTo() 方法

## 实例

使用淡出效果来隐藏一个 <p> 元素：

```
$(".btn1").click(function(){
  $("p").fadeTo(1000,0.4);
});
```

## 定义和用法

fadeTo() 方法将被选元素的不透明度逐渐地改变为指定的值。

### 语法

```
$(_selector_).fadeTo(_speed_,_opacity_,_callback_)
```

| 参数 | 描述 |
| --- | --- |
| *speed* | 可选。规定元素从当前透明度到指定透明度的速度。可能的值： 毫秒 （比如 1500） "slow" "normal" "fast" |
| *opacity* | 必需。规定要淡入或淡出的透明度。必须是介于 0.00 与 1.00 之间的数字。 |
| *callback* | 可选。fadeTo 函数执行完之后，要执行的函数。如需学习更多有关 callback 的内容，请访问我们的 jQuery Callback 这一章。除非设置了 speed 参数，否则不能设置该参数。 |

# jQuery 效果 - hide() 方法

## 实例

隐藏可见的 <p> 元素：

```
$(".btn1").click(function(){
  $("p").hide();
});
```

## 定义和用法

如果被选的元素已被显示，则隐藏该元素。

### 语法

```
$(_selector_).hide(_speed_,_callback_)
```

| 参数 | 描述 |
|---|---|
| *speed* | 可选。规定元素从可见到隐藏的速度。默认为 "0"。可能的值： 毫秒 （比如 1500） "slow" "normal" "fast"在设置速度的情况下，元素从可见到隐藏的过程中，会逐渐地改变其高度、宽度、外边距、内边距和透明度。 |
| *callback* | 可选。hide 函数执行完之后，要执行的函数。如需学习更多有关 callback 的内容，请访问我们的 jQuery Callback 这一章。除非设置了 speed 参数，否则不能设置该参数。 |

## 提示和注释

提示：如果元素已经是完全可见，则该效果不产生任何变化，除非规定了 callback 函数。

## 亲自试一试 - 实例

使用 speed 参数

使用 speed 参数来隐藏和显示元素。

使用 speed 和 callback 参数

使用 speed 和 callback 参数来隐藏和显示元素。

# jQuery 效果 - show() 方法

## 实例

显示出隐藏的 <p> 元素。

```
$(".btn2").click(function(){
  $("p").show();
});
```

## 定义和用法

如果被选元素已被隐藏，则显示这些元素：

### 语法

```
$(_selector_).show(_speed_,_callback_)
```

| 参数 | 描述 |
| --- | --- |
| *speed* | 可选。规定元素从隐藏到完全可见的速度。默认为 "0"。可能的值： 毫秒（比如 1500） "slow" "normal" "fast"在设置速度的情况下，元素从隐藏到完全可见的过程中，会逐渐地改变其高度、宽度、外边距、内边距和透明度。 |
| *callback* | 可选。show 函数执行完之后，要执行的函数。如需学习更多有关 callback 的内容，请访问我们的 jQuery Callback 这一章。除非设置了 speed 参数，否则不能设置该参数。 |

## 提示和注释

提示：如果元素已经是完全可见，则该效果不产生任何变化，除非规定了 callback 函数。

注释：该效果适用于通过 jQuery 隐藏的元素，或在 CSS 中声明 display:none 的元素（但不适用于 visibility:hidden 的元素）。

## 亲自试一试 - 实例

使用 speed 参数

使用 speed 参数来隐藏和显示元素。

## 使用 speed 和 callback 参数

使用 speed 和 callback 参数来隐藏和显示元素。

# jQuery 效果 - slideDown() 方法

## 实例

以滑动方式显示隐藏的 <p> 元素：

```
$(".btn2").click(function(){
  $("p").slideDown();
});
```

## 定义和用法

slideDown() 方法通过使用滑动效果，显示隐藏的被选元素。

### 语法

```
$(_selector_).slideDown(_speed_,_callback_)
```

| 参数 | 描述 |
| --- | --- |
| *speed* | 可选。规定元素从隐藏到可见的速度（或者相反）。默认为 "normal"。可能的值： 毫秒 （比如 1500） "slow" "normal" "fast"在设置速度的情况下，元素从隐藏到可见的过程中，会逐渐地改变其高度。 |
| *callback* | 可选。slideDown 函数执行完之后，要执行的函数。如需学习更多有关 callback 的内容，请访问我们的 jQuery Callback 这一章。除非设置了 speed 参数，否则不能设置该参数。 |

## 提示和注释

提示：如果元素已经是完全可见，则该效果不产生任何变化，除非规定了 callback 函数。

注释：该效果适用于通过 jQuery 隐藏的元素，或在 CSS 中声明 display:none 的元素（但不适用于 visibility:hidden 的元素）。

## 亲自试一试 - 实例

使用 speed 参数

使用 speed 参数来隐藏和显示元素。

# jQuery 效果 - slideToggle() 方法

## 实例

通过使用滑动效果，在显示和隐藏状态之间切换 <p> 元素：

```
$(".btn1").click(function(){
  $("p").slideToggle();
});
```

## 定义和用法

slideToggle() 方法通过使用滑动效果（高度变化）来切换元素的可见状态。

如果被选元素是可见的，则隐藏这些元素，如果被选元素是隐藏的，则显示这些元素。

### 语法

```
$(_selector_).slideToggle(_speed_,_callback_)
```

| 参数 | 描述 |
| --- | --- |
| *speed* | 可选。规定元素从隐藏到可见的速度（或者相反）。默认为 "normal"。可能的值： 毫秒 （比如 1500） "slow" "normal" "fast"在设置速度的情况下，元素在切换的过程中，会逐渐地改变其高度（这样会创造滑动效果）。 |
| *callback* | 可选。toggle 函数执行完之后，要执行的函数。如需学习更多有关 callback 的内容，请访问我们的 jQuery Callback 这一章。除非设置了 speed 参数，否则不能设置该参数。 |

## 提示和注释

提示：如果元素已经隐藏，则该效果不产生任何变化，除非规定了 callback 函数。

## 亲自试一试 - 实例

使用 speed 参数

使用 speed 参数来切换上下滑动地显示和隐藏元素。

# jQuery 效果 - slideUp() 方法

## 实例

以滑动方式隐藏 <p> 元素：

```
$(".btn1").click(function(){
  $("p").slideUp();
});
```

## 定义和用法

通过使用滑动效果，隐藏被选元素，如果元素已显示出来的话。

### 语法

```
$(_selector_).slideUp(_speed_,_callback_)
```

| 参数 | 描述 |
|---|---|
| *speed* | 可选。规定元素从可见到隐藏的速度（或者相反）。默认为 "normal"。可能的值： 毫秒 （比如 1500） "slow" "normal" "fast"在设置速度的情况下，元素从可见到隐藏的过程中，会逐渐地改变其高度（这样会创造滑动效果）。 |
| *callback* | 可选。slideUp 函数执行完之后，要执行的函数。如需学习更多有关 callback 的内容，请访问我们的 jQuery Callback 这一章。除非设置了 speed 参数，否则不能设置该参数。 |

## 提示和注释

提示：如果元素已经隐藏，则该效果不产生任何变化，除非规定了 callback 函数。

## 亲自试一试 - 实例

使用 speed 参数

使用 speed 参数来隐藏和显示元素。

# jQuery 效果 - stop() 方法

## 实例

停止当前正在运行的动画：

```
$("#stop").click(function(){
  $("#box").stop();
});
```

## 定义和用法

stop() 方法停止当前正在运行的动画。

### 语法

```
$(_selector_).stop(_stopAll_,_goToEnd_)
```

| 参数 | 描述 |
| --- | --- |
| *stopAll* | 可选。规定是否停止被选元素的所有加入队列的动画。 |
| *goToEnd* | 可选。规定是否允许完成当前的动画。该参数只能在设置了 stopAll 参数时使用。 |

## 亲自试一试 - 实例

停止动画队列

停止被选元素的所有加入队列的动画。

在当前动画完成后停止动画队列

停止被选元素的所有加入队列的动画，但允许完成当前动画。

# jQuery 效果 - toggle() 方法

## 实例

切换 <p> 元素的显示与隐藏状态：

```
$(".btn1").click(function(){
  $("p").hide();
});
```

## 定义和用法

toggle() 方法切换元素的可见状态。

如果被选元素可见，则隐藏这些元素，如果被选元素隐藏，则显示这些元素。

### 语法

```
$(_selector_).toggle(_speed_,_callback_,_switch_)
```

| 参数 | 描述 |
| --- | --- |
| *speed* | 可选。规定元素从可见到隐藏的速度（或者相反）。默认为 "0"。可能的值：毫秒 （比如 1500） "slow" "normal" "fast"在设置速度的情况下，元素从可见到隐藏的过程中，会逐渐地改变其高度、宽度、外边距、内边距和透明度。如果设置此参数，则无法使用 switch 参数。 |
| *callback* | 可选。toggle 函数执行完之后，要执行的函数。如需学习更多有关 callback 的内容，请访问我们的 jQuery Callback 这一章。除非设置了 speed 参数，否则不能设置该参数。 |
| *switch* | 可选。布尔值。规定 toggle 是否隐藏或显示所有被选元素。 True - 显示所有元素 False - 隐藏所有元素如果设置此参数，则无法使用 speed 和 callback 参数。 |

## 提示和注释

注释：该效果适用于通过 jQuery 隐藏的元素，或在 CSS 中声明 display:none 的元素（但不适用于 visibility:hidden 的元素）。

## 亲自试一试 - 实例

### 使用 speed 参数

使用 speed 参数来隐藏和显示元素。

### 使用 switch 参数

使用 switch 参数来显示所有隐藏的段落。

# jQuery 参考手册 - 文档操作

## jQuery 文档操作方法

这些方法对于 XML 文档和 HTML 文档均是适用的，除了：html()。

| 方法 | 描述 |
| --- | --- |
| addClass() | 向匹配的元素添加指定的类名。 |
| after() | 在匹配的元素之后插入内容。 |
| append() | 向匹配元素集合中的每个元素结尾插入由参数指定的内容。 |
| appendTo() | 向目标结尾插入匹配元素集合中的每个元素。 |
| attr() | 设置或返回匹配元素的属性和值。 |
| before() | 在每个匹配的元素之前插入内容。 |
| clone() | 创建匹配元素集合的副本。 |
| detach() | 从 DOM 中移除匹配元素集合。 |
| empty() | 删除匹配的元素集合中所有的子节点。 |
| hasClass() | 检查匹配的元素是否拥有指定的类。 |
| html() | 设置或返回匹配的元素集合中的 HTML 内容。 |
| insertAfter() | 把匹配的元素插入到另一个指定的元素集合的后面。 |
| insertBefore() | 把匹配的元素插入到另一个指定的元素集合的前面。 |
| prepend() | 向匹配元素集合中的每个元素开头插入由参数指定的内容。 |
| prependTo() | 向目标开头插入匹配元素集合中的每个元素。 |
| remove() | 移除所有匹配的元素。 |
| removeAttr() | 从所有匹配的元素中移除指定的属性。 |
| removeClass() | 从所有匹配的元素中删除全部或者指定的类。 |
| replaceAll() | 用匹配的元素替换所有匹配到的元素。 |
| replaceWith() | 用新内容替换匹配的元素。 |
| text() | 设置或返回匹配元素的内容。 |
| toggleClass() | 从匹配的元素中添加或删除一个类。 |
| unwrap() | 移除并替换指定元素的父元素。 |
| val() | 设置或返回匹配元素的值。 |
| wrap() | 把匹配的元素用指定的内容或元素包裹起来。 |
| wrapAll() | 把所有匹配的元素用指定的内容或元素包裹起来。 |
| wrapinner() | 将每一个匹配的元素的子内容用指定的内容或元素包裹起来。 |

# jQuery 属性操作 - addClass() 方法

## 实例

向第一个 p 元素添加一个类：

```
$("button").click(function(){
  $("p:first").addClass("intro");
});
```

## 定义和用法

addClass() 方法向被选元素添加一个或多个类。

该方法不会移除已存在的 class 属性，仅仅添加一个或多个 class 属性。

提示：如需添加多个类，请使用空格分隔类名。

### 语法

```
$(selector).addClass(_class_)
```

| 参数 | 描述 |
| --- | --- |
| class | 必需。规定一个或多个 class 名称。 |

## 使用函数来添加类

使用函数向被选元素添加类。

### 语法

```
$(selector).addClass(_function_(_index_,_oldclass)_)
```

| 参数 | 描述 |
| --- | --- |
| *function(index,oldclass)* | 必需。规定返回一个或多个待添加类名的函数。 *index* - 可选。选择器的 index 位置。 *class* - 可选。选择器的旧的类名。 |

# 亲自试一试 - 实例

向元素添加两个类

如何向被选元素添加两个 class。

改变元素的类

如任何使用 addClass() 和 removeClass() 来移除 class，并添加新的 class。

# jQuery 文档操作 - after() 方法

## 实例

在每个 p 元素后插入内容：

```
$("button").click(function(){
  $("p").after("<p>Hello world!</p>");
});
```

## 定义和用法

after() 方法在被选元素后插入指定的内容。

### 语法

```
$(selector).after(_content_)
```

| 参数 | 描述 |
|---|---|
| content | 必需。规定要插入的内容（可包含 HTML 标签）。 |

## 使用函数来插入内容

使用函数在被选元素之后插入指定的内容。

### 语法

```
$(selector).after(_function_(_index_))
```

| 参数 | 描述 |
|---|---|
| *function*(*index*) | 必需。规定返回待插入内容的函数。 *index* - 可选。接收选择器的 index 位置。 |

# jQuery 文档操作 - append() 方法

## 实例

在每个 p 元素结尾插入内容：

```
$("button").click(function(){
  $("p").append(" <b>Hello world!</b>");
});
```

## 定义和用法

append() 方法在被选元素的结尾（仍然在内部）插入指定内容。

提示：append() 和 appendTo() 方法执行的任务相同。不同之处在于：内容的位置和选择器。

### 语法

```
$(selector).append(_content_)
```

| 参数 | 描述 |
|---|---|
| content | 必需。规定要插入的内容（可包含 HTML 标签）。 |

## 使用函数来附加内容

使用函数在指定元素的结尾插入内容。

### 语法

```
$(selector).append(_function(index,html)_)
```

| 参数 | 描述 |
|---|---|
| *function(index,html)* | 必需。规定返回待插入内容的函数。 *index* - 可选。接收选择器的 index 位置。 *html* - 可选。接收选择器的当前 HTML。 |

# jQuery 文档操作 - appendTo() 方法

## 实例

在每个 p 元素结尾插入内容：

```
$("button").click(function(){
  $("<b>Hello World!</b>").appendTo("p");
});
```

## 定义和用法

appendTo() 方法在被选元素的结尾（仍然在内部）插入指定内容。

提示：append() 和 appendTo() 方法执行的任务相同。不同之处在于：内容和选择器的位置，以及 append() 能够使用函数来附加内容。

## 语法

```
$(_content_).appendTo(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *content* | 必需。规定要插入的内容（可包含 HTML 标签）。 |
| *selector* | 必需。规定把内容追加到哪个元素上。 |

# jQuery 属性操作 - attr() 方法

## 实例

改变图像的 width 属性：

```
$("button").click(function(){
  $("img").attr("width","180");
});
```

## 定义和用法

attr() 方法设置或返回被选元素的属性值。

根据该方法不同的参数，其工作方式也有所差异。

## 返回属性值

返回被选元素的属性值。

### 语法

```
$(_selector_).attr(_attribute_)
```

| 参数 | 描述 |
| --- | --- |
| *attribute* | 规定要获取其值的属性。 |

## 设置属性/值

设置被选元素的属性和值。

### 语法

```
$(_selector_).attr(_attribute_,_value_)
```

| 参数 | 描述 |
| --- | --- |
| *attribute* | 规定属性的名称。 |
| *value* | 规定属性的值。 |

# 使用函数来设置属性/值

设置被选元素的属性和值。

## 语法

```
$(_selector_).attr(_attribute_,_function(index,oldvalue)_)
```

| 参数 | 描述 |
| --- | --- |
| *attribute* | 规定属性的名称。 |
| *function(index,oldvalue)* | 规定返回属性值的函数。该函数可接收并使用选择器的 index 值和当前属性值。 |

# 设置多个属性/值对

为被选元素设置一个以上的属性和值。

## 语法

```
$(_selector_).attr({_attribute:value_, _attribute:value_ ...})
```

| 参数 | 描述 |
| --- | --- |
| *attribute:value* | 规定一个或多个属性/值对。 |

# jQuery 文档操作 - before() 方法

## 实例

在每个 p 元素前插入内容：

```
$("button").click(function(){
  $("p").before("<p>Hello world!</p>");
});
```

## 定义和用法

before() 方法在被选元素前插入指定的内容。

### 语法

```
$(selector).before(_content_)
```

| 参数 | 描述 |
| --- | --- |
| content | 必需。规定要插入的内容（可包含 HTML 标签）。 |

## 使用函数来插入内容

使用函数在指定的元素前面插入内容。

### 语法

```
$(selector).before(_function(index)_)
```

| 参数 | 描述 |
| --- | --- |
| *function(index)* | 必需。规定返回待插入内容的函数。 *index* - 可选。接收选择器的 index 位置。 |

# jQuery 文档操作 - clone() 方法

## 实例

克隆并追加一个 p 元素：

```
$("button").click(function(){
  $("body").append($("p").clone());
});
```

## 定义和用法

clone() 方法生成被选元素的副本，包含子节点、文本和属性。

## 语法

```
$(_selector_).clone(_includeEvents_)
```

| 参数 | 描述 |
|---|---|
| *includeEvents* | 可选。布尔值。规定是否复制元素的所有事件处理。默认地，副本中不包含事件处理器。 |

## 亲自试一试 - 实例

复制一个元素，包括事件处理器

使用 clone() 方法来复制元素，包括其事件处理器。

# jQuery 文档操作 - detach() 方法

## 实例

移除所有 p 元素：

```
$("button").click(function(){
  $("p").detach();
});
```

## 定义和用法

detach() 方法移除被选元素，包括所有文本和子节点。

这个方法会保留 jQuery 对象中的匹配的元素，因而可以在将来再使用这些匹配的元素。

detach() 会保留所有绑定的事件、附加的数据，这一点与 remove() 不同。

### 语法

```
$(_selector_).detach()
```

## 亲自试一试 - 实例

移动元素

使用 detach() 方法来移动元素。

删除并恢复一个元素

使用 detach() 方法来删除并恢复一个元素。

移动元素并保留其 click 事件

使用 detach() 方法来移动元素，并保留元素的 jQuery 数据。

# jQuery 文档操作 - empty() 方法

## 实例

移除 p 元素的内容：

```
$(".btn1").click(function(){
  $("p").empty();
});
```

## 定义和用法

empty() 方法从被选元素移除所有内容，包括所有文本和子节点。

### 语法

```
$(_selector_).empty()
```

## 亲自试一试 - 实例

移除元素的内容

使用 empty() 方法从元素移除内容。

# jQuery 属性操作 - hasClass() 方法

## 实例

检查第一个 <p> 元素是否包含 "intro" 类：

```
$("button").click(function(){
  alert($("p:first").hasClass("intro"));
});
```

## 定义和用法

hasClass() 方法检查被选元素是否包含指定的 class。

### 语法

```
$(_selector_).hasClass(_class_)
```

| 参数 | 描述 |
| --- | --- |
| class | 必需。规定需要在指定元素中查找的类。 |

# jQuery 文档操作 - html() 方法

## 实例

设置所有 p 元素的内容：

```
$(".btn1").click(function(){
  $("p").html("Hello <b>world</b>!");
});
```

## 定义和用法

html() 方法返回或设置被选元素的内容 (inner HTML)。

如果该方法未设置参数，则返回被选元素的当前内容。

## 返回元素内容

当使用该方法返回一个值时，它会返回第一个匹配元素的内容。

### 语法

```
$(selector).html()
```

## 设置元素内容

当使用该方法设置一个值时，它会覆盖所有匹配元素的内容。

### 语法

```
$(_selector_).html(_content_)
```

| 参数 | 描述 |
| --- | --- |
| *content* | 可选。规定被选元素的新内容。该参数可包含 HTML 标签。 |

## 使用函数来设置元素内容

使用函数来设置所有匹配元素的内容。

## 语法

```
$(_selector_).html(_function(index,oldcontent)_)
```

| 参数 | 描述 |
| --- | --- |
| *function(index,oldcontent)* | 规定一个返回被选元素的新内容的函数。 *index* - 可选。接收选择器的 index 位置。 *oldcontent* - 可选。接收选择器的当前内容。 |

# jQuery 文档操作 - insertAfter() 方法

## 实例

在每个 p 元素之后插入 span 元素：

```
$("button").click(function(){
  $("<span>Hello world!</span>").insertAfter("p");
});
```

## 定义和用法

insertAfter() 方法在被选元素之后插入 HTML 标记或已有的元素。

注释：如果该方法用于已有元素，这些元素会被从当前位置移走，然后被添加到被选元素之后。

### 语法

```
$(_content_).insertAfter(_selector_)
```

| 参数 | 描述 |
|---|---|
| *content* | 必需。规定要插入的内容。可能的值：选择器表达式 HTML 标记 |
| *selector* | 必需。规定在何处插入被选元素。 |

## 亲自试一试 - 实例

插入已有的元素

如何使用 insertAfter() 方法在每个被选元素之后插入已存在的元素。

# jQuery 文档操作 - insertBefore() 方法

## 实例

在每个 p 元素之前插入 span 元素：

```
$("button").click(function(){
  $("<span>Hello world!</span>").insertBefore("p");
});
```

## 定义和用法

insertBefore() 方法在被选元素之前插入 HTML 标记或已有的元素。

注释：如果该方法用于已有元素，这些元素会被从当前位置移走，然后被添加到被选元素之前。

### 语法

```
$(_content_).insertBefore(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *content* | 必需。规定要插入的内容。可能的值：选择器表达式 HTML 标记 |
| *selector* | 必需。规定在何处插入被选元素。 |

## 亲自试一试 - 实例

插入已有的元素

如何使用 insertAfter() 方法在每个被选元素之前插入已存在的元素。

# jQuery 文档操作 - prepend() 方法

## 实例

在 p 元素的开头插入内容：

```
$(".btn1").click(function(){
  $("p").prepend("<b>Hello world!</b>");
});
```

## 定义和用法

prepend() 方法在被选元素的开头（仍位于内部）插入指定内容。

提示：prepend() 和 prependTo() 方法作用相同。差异在于语法：内容和选择器的位置，以及 prependTo() 无法使用函数来插入内容。

### 语法

```
$(_selector_).prepend(_content_)
```

| 参数 | 描述 |
| --- | --- |
| content | 必需。规定要插入的内容（可包含 HTML 标签）。 |

## 使用函数来附加内容

使用函数在被选元素的开头插入指定的内容。

### 语法

```
$(_selector_).prepend(_function(index,html)_)
```

| 参数 | 描述 |
| --- | --- |
| *function(index,html)* | 必需。规定返回待插入内容的函数。 `index` - 可选。接受选择器的 index 位置。 `html` - 可选。接受选择器的当前 HTML。 |

# jQuery 文档操作 - prependTo() 方法

## 实例

在每个 p 元素的开头插入内容：

```
$(".btn1").click(function(){
  $("<b>Hello World!</b>").prependTo("p");
});
```

## 定义和用法

prependTo() 方法在被选元素的开头（仍位于内部）插入指定内容。

提示：prepend() 和 prependTo() 方法作用相同。差异在于语法：内容和选择器的位置，以及 prepend() 能够使用函数来插入内容。

### 语法

```
$(_content_).prependTo(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *content* | 必需。规定要插入的内容（可包含 HTML 标签）。 |
| *selector* | 必需。规定在何处插入内容。 |

# jQuery 文档操作 - remove() 方法

## 实例

移除所有 <p> 元素：

```
$("button").click(function(){
  $("p").remove();
});
```

## 定义和用法

remove() 方法移除被选元素，包括所有文本和子节点。

该方法不会把匹配的元素从 jQuery 对象中删除，因而可以在将来再使用这些匹配的元素。

但除了这个元素本身得以保留之外，remove() 不会保留元素的 jQuery 数据。其他的比如绑定的事件、附加的数据等都会被移除。这一点与 detach() 不同。

### 语法

```
$(_selector_).remove()
```

## 亲自试一试 - 实例

移动元素

使用 remove() 方法来移动元素。

# jQuery 属性操作 - removeAttr() 方法

## 实例

从任何 p 元素中移除 id 属性：

```
$("button").click(function(){
  $("p").removeAttr("id");
});
```

## 定义和用法

removeAttr() 方法从被选元素中移除属性。

### 语法

```
$(_selector_).removeAttr(_attribute_)
```

| 参数 | 描述 |
| --- | --- |
| *attribute* | 必需。规定从指定元素中移除的属性。 |

# jQuery 属性操作 - removeClass() 方法

## 实例

移除所有 <p> 的 "intro" 类：

```
$("button").click(function(){
  $("p:first").removeClass("intro");
});
```

## 定义和用法

removeClass() 方法从被选元素移除一个或多个类。

注释：如果没有规定参数，则该方法将从被选元素中删除所有类。

### 语法

```
$(_selector_).removeClass(_class_)
```

| 参数 | 描述 |
| --- | --- |
| *class* | 可选。规定要移除的 class 的名称。如需移除若干类，请使用空格来分隔类名。如果不设置该参数，则会移除所有类。 |

## 使用函数来移除类

使用函数来删除被选元素中的类。

```
$(_selector_).removeClass(_function(index,oldclass)_)
```

| 参数 | 描述 |
| --- | --- |
| *function(index,oldclass)* | 必需。通过运行函数来移除指定的类。 `index` - 可选。接受选择器的 index 位置。 `html` - 可选。接受选择器的旧的类值。 |

## 亲自试一试 - 实例

### 改变元素的类

如何使用 addClass() 和 removeClass() 来移除一个类，并添加一个新的类。

# jQuery 文档操作 - replaceAll() 方法

## 实例

用粗体文本替换每个段落：

```
$(".btn1").click(function(){
   $("p").replaceAll("<b>Hello world!</b>");
});
```

## 定义和用法

replaceAll() 方法用指定的 HTML 内容或元素替换被选元素。

提示：replaceAll() 与 replaceWith() 作用相同。差异在于语法：内容和选择器的位置，以及 replaceWith() 能够使用函数进行替换。

### 语法

```
$(_content_).replaceAll(_selector_)
```

| 参数 | 描述 |
|---|---|
| *content* | 必需。规定替换被选元素的内容。可能的值: `HTML 代码` - 比如 ("<div></div>") `新元素` - 比如 (document.createElement("div")) `已存在的元素` - 比如 ($(".div1")) 已存在的元素不会被移动，只会被复制，并包裹被选元素。 |
| *selector* | 必需。规定要替换的元素。 |

## 亲自试一试 - 实例

使用新元素来替换元素

使用 document.createElement() 来创建一个新的 DOM 元素，然后用它替换被选元素。

# jQuery 文档操作 - replaceWith() 方法

## 实例

用粗体文本替换每个段落：

```
$(".btn1").click(function(){
   $("p").replaceWith("<b>Hello world!</b>");
});
```

## 定义和用法

replaceWith() 方法用指定的 HTML 内容或元素替换被选元素。

提示：replaceWith() 与 replaceAll() 作用相同。差异在于语法：内容和选择器的位置，以及 replaceAll() 无法使用函数进行替换。

### 语法

```
$(_selector_).replaceWith(_content_)
```

| 参数 | 描述 |
| --- | --- |
| *content* | 必需。规定替换被选元素的内容。可能的值: `HTML 代码` - 比如 ("<div></div>") `新元素` - 比如 (document.createElement("div")) `已存在的元素` - 比如 ($(".div1")) 已存在的元素不会被移动，只会被复制，并包裹被选元素。 |
| *selector* | 必需。规定要替换的元素。 |

## 使用函数来替换元素

使用函数把被选元素替换为新内容。

### 语法

```
$(_selector_).replaceWith(_function()_)
```

| 参数 | 描述 |
| --- | --- |
| *function()* | 必需。返回待替换被选元素的新内容的函数。 |

## 亲自试一试 - 实例

使用新元素来替换元素

使用 document.createElement() 来创建一个新的 DOM 元素，然后用它替换被选元素。

# jQuery 文档操作 - text() 方法

## 实例

设置所有 <p> 元素的内容：

```
$(".btn1").click(function(){
  $("p").text("Hello <b>world</b>!");
});
```

## 定义和用法

text() 方法方法设置或返回被选元素的文本内容。

## 返回文本内容

当该方法用于返回一个值时，它会返回所有匹配元素的组合的文本内容（会删除 HTML 标记）。

### 语法

```
$(_selector_).text()
```

## 设置文本内容

当该方法用于设置值时，它会覆盖被选元素的所有内容。

```
$(_selector_).text(_content_)
```

| 参数 | 描述 |
| --- | --- |
| *content* | 规定被选元素的新文本内容。注释：特殊字符会被编码。 |

## 使用函数设置文本内容

使用函数设置所有被选元素的文本内容。

## 语法

```
$(_selector_).text(_function(index,oldcontent)_)
```

| 参数 | 描述 |
| --- | --- |
| *function(index,oldcontent)* | 必需。规定返回被选元素的新文本内容的函数。 `index` - 可选。接受选择器的 index 位置。 `html` - 可选。接受选择器的当前内容。 |

# jQuery 属性操作 - toggleClass() 方法

## 实例

对设置和移除所有 <p> 元素的 "main" 类进行切换：

```
$("button").click(function(){
  $("p").toggleClass("main");
});
```

## 定义和用法

toggleClass() 对设置或移除被选元素的一个或多个类进行切换。

该方法检查每个元素中指定的类。如果不存在则添加类，如果已设置则删除之。这就是所谓的切换效果。

不过，通过使用 "switch" 参数，您能够规定只删除或只添加类。

### 语法

```
$(_selector_).toggleClass(_class_,_switch_)
```

| 参数 | 描述 |
|---|---|
| *class* | 必需。规定添加或移除 class 的指定元素。如需规定若干 class，请使用空格来分隔类名。 |
| *switch* | 可选。布尔值。规定是否添加或移除 class。 |

## 使用函数来切换类

```
$(_selector_).toggleClass(_function(index,class)_,_switch_)
```

| 参数 | 描述 |
|---|---|
| *function(index,class)* | 必需。规定返回需要添加或删除的一个或多个类名的函数。 `index` - 可选。接受选择器的 index 位置。 `class` - 可选。接受选择器的当前的类。 |
| *switch* | 可选。布尔值。规定是否添加(true)或移除(false)类。 |

# jQuery 文档操作 - unwrap() 方法

## 实例

删除所有 <p> 元素的父元素：

```
$("button").click(function(){
  $("p").unwrap();
});
```

## 定义和用法

unwrap() 方法删除被选元素的父元素。

### 语法

```
$(_selector_).unwrap()
```

## 亲自试一试 - 实例

包裹或解开元素

对包裹和解开元素的任务进行切换。

# jQuery 属性操作 - val() 方法

## 实例

设置输入域的值：

```
$("button").click(function(){
  $(":text").val("Hello World");
});
```

## 定义和用法

val() 方法返回或设置被选元素的值。

元素的值是通过 value 属性设置的。该方法大多用于 input 元素。

如果该方法未设置参数，则返回被选元素的当前值。

### 语法

```
$(_selector_).val(_value_)
```

| 参数 | 描述 |
| --- | --- |
| *value* | 可选。规定被选元素的新内容。 |

## 返回 Value 属性

返回第一个匹配元素的 value 属性的值。

### 语法

```
$(_selector_).val()
```

## 设置 Value 属性的值

```
$(_selector_).val(_value_)
```

| 参数 | 描述 |
| --- | --- |
| *value* | 设置 Value 属性的值。 |

## 使用函数设置 Value 属性的值

```
$(_selector_).val(_function(index,oldvalue)_)
```

| 参数 | 描述 |
| --- | --- |
| *function(index,oldvalue)* | 规定返回要设置的值的函数。 *index* - 可选。接受选择器的 index 位置。 *oldvalue* - 可选。接受选择器的当前 Value 属性。 |

# jQuery 文档操作 - wrap() 方法

## 实例

在 <div> 元素中包裹每个段落：

```
$(".btn1").click(function(){
   $("p").wrap("<div></div>");
});
```

## 定义和用法

wrap() 方法把每个被选元素放置在指定的 HTML 内容或元素中。

### 语法

```
$(_selector_).wrap(_wrapper_)
```

| 参数 | 描述 |
| --- | --- |
| *wrapper* | 必需。规定包裹被选元素的内容。可能的值: `HTML 代码` - 比如 ("<div></div>") `新元素` - 比如 (document.createElement("div")) `已存在的元素` - 比如 ($(".div1")) 已存在的元素不会被移动，只会被复制，并包裹被选元素。 |

## 使用函数来包裹元素

使用函数来规定在每个被选元素周围包裹的内容。

### 语法

```
$(_selector_).wrap(_function()_)
```

| 参数 | 描述 |
| --- | --- |
| *function()* | 必需。规定返回包裹元素的函数。 |

## 亲自试一试 - 实例

### 使用新元素来包裹

创建一个新的 DOM 元素来包裹每个被选元素。

### 包裹或解开元素

对包裹和解开元素的任务进行切换。

# jQuery 文档操作 - wrapAll() 方法

## 实例

在 <div> 中包裹所有段落：

```
$(".btn1").click(function(){
   $("p").wrapAll("<div></div>");
});
```

## 定义和用法

wrapAll() 在指定的 HTML 内容或元素中放置所有被选的元素。

### 语法

```
$(_selector_).wrapAll(_wrapper_)
```

| 参数 | 描述 |
| --- | --- |
| wrapper | 必需。规定包裹被选元素的内容。可能的值: `HTML 代码` - 比如 ("<div></div>") `新的 DOM 元素` - 比如 (document.createElement("div")) `已存在的元素` - 比如 ($(".div1")) 已存在的元素不会被移动，只会被复制，并包裹被选元素。 |

## 亲自试一试 - 实例

使用新元素来包裹

创建一个新的 DOM 元素来包裹每个被选元素。

# jQuery 文档操作 - wrapInner() 方法

## 实例

在每个 p 元素的内容上包围 b 元素：

```
$(".btn1").click(function(){
   $("p").wrapInner("<b></b>");
});
```

## 定义和用法

wrapInner() 方法使用指定的 HTML 内容或元素，来包裹每个被选元素中的所有内容 (inner HTML)。

### 语法

```
$(_selector_).wrapInner(_wrapper_)
```

| 参数 | 描述 |
| --- | --- |
| *wrapper* | 必需。规定包围在被选元素的内容周围的内容。可能的值: `HTML 代码` - 比如 ("<div></div>") `新的 DOM 元素` - 比如 (document.createElement("div")) `已存在的元素` - 比如 ($(".div1")) 已存在的元素不会被移动，只会被复制，并包裹被选元素。 |

## 使用函数包裹内容

使用函数来规定包围在每个被选元素周围的内容。

### 语法

```
$(_selector_).wrapInner(_function()_)
```

| 参数 | 描述 |
| --- | --- |
| *function()* | 必需。规定返回包围元素的函数。 |

# 亲自试一试 - 实例

使用新元素来包裹

创建一个新的 DOM 元素来包裹每个被选元素。

# jQuery 参考手册 - 属性操作

## jQuery 属性操作方法

下面列出的这些方法获得或设置元素的 DOM 属性。

这些方法对于 XML 文档和 HTML 文档均是适用的，除了：html()。

| 方法 | 描述 |
| --- | --- |
| addClass() | 向匹配的元素添加指定的类名。 |
| attr() | 设置或返回匹配元素的属性和值。 |
| hasClass() | 检查匹配的元素是否拥有指定的类。 |
| html() | 设置或返回匹配的元素集合中的 HTML 内容。 |
| removeAttr() | 从所有匹配的元素中移除指定的属性。 |
| removeClass() | 从所有匹配的元素中删除全部或者指定的类。 |
| toggleClass() | 从匹配的元素中添加或删除一个类。 |
| val() | 设置或返回匹配元素的值。 |

注释：中也列出了以上方法。本参考页的作用是方便用户单独查阅有关属性操作方面的方法。

# jQuery 参考手册 - CSS 操作

## jQuery CSS 操作函数

下面列出的这些方法设置或返回元素的 CSS 相关属性。

| CSS 属性 | 描述 |
| --- | --- |
| css() | 设置或返回匹配元素的样式属性。 |
| height() | 设置或返回匹配元素的高度。 |
| offset() | 返回第一个匹配元素相对于文档的位置。 |
| offsetParent() | 返回最近的定位祖先元素。 |
| position() | 返回第一个匹配元素相对于父元素的位置。 |
| scrollLeft() | 设置或返回匹配元素相对滚动条左侧的偏移。 |
| scrollTop() | 设置或返回匹配元素相对滚动条顶部的偏移。 |
| width() | 设置或返回匹配元素的宽度。 |

## 参阅

教程：CSS 教程

参考手册：<a href="">CSS 参考手册</a>

# jQuery CSS 操作 - css() 方法

## 实例

设置 <p> 元素的颜色：

```
$(".btn1").click(function(){
  $("p").css("color","red");
});
```

## 定义和用法

css() 方法返回或设置匹配的元素的一个或多个样式属性。

## 返回 CSS 属性值

返回第一个匹配元素的 CSS 属性值。

注释：当用于返回一个值时，不支持简写的 CSS 属性（比如 "background" 和 "border"）。

```
$(_selector_).css(_name_)
```

| 参数 | 描述 |
|---|---|
| *name* | 必需。规定 CSS 属性的名称。该参数可包含任何 CSS 属性。比如 "color"。 |

### 实例

取得第一个段落的 color 样式属性的值：

```
$("p").css("color");
```

## 设置 CSS 属性

设置所有匹配元素的指定 CSS 属性。

```
$(_selector_).css(_name_,_value_)
```

| 参数 | 描述 |
| --- | --- |
| *name* | 必需。规定 CSS 属性的名称。该参数可包含任何 CSS 属性，比如 "color"。 |
| *value* | 可选。规定 CSS 属性的值。该参数可包含任何 CSS 属性值，比如 "red"。如果设置了空字符串值，则从元素中删除指定属性。 |

## 实例

将所有段落的颜色设为红色：

```
$("p").css("color","red");
```

# 使用函数来设置 CSS 属性

设置所有匹配的元素中样式属性的值。

此函数返回要设置的属性值。接受两个参数，index 为元素在对象集合中的索引位置，value 是原先的属性值。

```
$(_selector_).css(_name_,_function(index,value)_)
```

| 参数 | 描述 |
| --- | --- |
| *name* | 必需。规定 CSS 属性的名称。该参数可包含任何 CSS 属性，比如 "color"。 |
| *function(index,value)* | 规定返回 CSS 属性新值的函数。 `index` - 可选。接受选择器的 index 位置 `oldvalue` - 可选。接受 CSS 属性的当前值。 |

## 实例 1

将所有段落的颜色设为红色：

```
$("button").click(function(){
    $("p").css("color",function(){return "red";});
    });
```

## 实例 2

逐渐增加 div 的宽度：

```
$("div").click(function() {
  $(this).css(
    "width", function(index, value) {return parseFloat(value) * 1.2;}
  );
});
```

# 设置多个 CSS 属性/值对

```
$(_selector_).css({_property:value_, _property:value_, ...})
```

把“名/值对”对象设置为所有匹配元素的样式属性。

这是一种在所有匹配的元素上设置大量样式属性的最佳方式。

| 参数 | 描述 |
| --- | --- |
| *{property:value}* | 必需。规定要设置为样式属性的“名称/值对”对象。该参数可包含若干对 CSS 属性名称/值。比如 {"color":"red","font-weight":"bold"} |

## 实例

```
$("p").css({
  "color":"white",
  "background-color":"#98bf21",
  "font-family":"Arial",
  "font-size":"20px",
  "padding":"5px"
  });
```

# jQuery CSS 操作 - height() 方法

## 实例

设置 <p> 元素的高度：

```
$(".btn1").click(function(){
  $("p").height(50);
});
```

## 定义和用法

height() 方法返回或设置匹配元素的高度。

## 返回高度

返回第一个匹配元素的高度。

如果不为该方法设置参数，则返回以像素计的匹配元素的高度。

### 语法

```
$(_selector_).height()
```

## 设置高度

设置所有匹配元素的高度。

### 语法

```
$(_selector_).height(_length_)
```

| 参数 | 描述 |
| --- | --- |
| *length* | 可选。规定元素的高度。如果没有规定长度单位，则使用默认的 px 单位。 |

## 使用函数来设置高度

使用函数来设置所有匹配元素的高度。

## 语法

```
$(_selector_).height(_function(index,oldheight)_)
```

| 参数 | 描述 |
| --- | --- |
| *function(index,oldheight)* | 规定返回被选元素新高度的函数。 `index` - 可选。接受选择器的 index 位置 `oldvalue` - 可选。接受选择器的当前值。 |

# 亲自试一试 - 实例

获得文档和窗口元素的高度

使用 height() 方法来获得 document 和 window 元素的当前高度。

使用 em 和 % 值来设置高度

使用指定的长度单位来设置元素的高度。

# jQuery CSS 操作 - offset() 方法

## 实例

获得 <p> 元素当前的偏移：

```
$(".btn1").click(function(){
  x=$("p").offset();
  $("#span1").text(x.left);
  $("#span2").text(x.top);
});
```

## 定义和用法

offset() 方法返回或设置匹配元素相对于文档的偏移（位置）。

## 返回偏移坐标

返回第一个匹配元素的偏移坐标。

该方法返回的对象包含两个整型属性：top 和 left，以像素计。此方法只对可见元素有效。

### 语法

```
$(_selector_).offset()
```

## 设置偏移坐标

设置所有匹配元素的偏移坐标。

### 语法

```
$(_selector_).offset(_value_)
```

| 参数 | 描述 |
| --- | --- |
| *value* | 必需。规定以像素计的 top 和 left 坐标。可能的值： 值对，比如 {top:100,left:0}； 带有 top 和 left 属性的对象。 |

# 使用函数来设置偏移坐标

使用函数来设置所有匹配元素的偏移坐标。

## 语法

```
$(_selector_).offset(_function(index,oldoffset)_)
```

| 参数 | 描述 |
|---|---|
| *function(index,oldoffset)* | 规定返回被选元素新偏移坐标的函数。 `index` - 可选。接受选择器的 index 位置 `oldvalue` - 可选。接受选择器的当前坐标。 |

# 亲自试一试 - 实例

使用对象来为对象设置新的 offset 值

使用新对象中的坐标来定位元素。

使用另一个元素的位置来为元素设置新的 offset 值

使用已有对象的位置来定位元素。

# jQuery CSS 操作 - offsetParent() 方法

## 实例

设置最近的祖先定位元素的背景颜色：

```
$("button").click(function(){
  $("p").offsetParent().css("background-color","red");
});
```

## 定义和用法

offsetParent() 方法返回最近的祖先定位元素。

定位元素指的是元素的 CSS position 属性被设置为 relative、absolute 或 fixed 的元素。

可以通过 jQuery 设置 position，或者通过 CSS 的 position 属性。

### 语法

```
$(_selector_).offsetParent()
```

# jQuery CSS 操作 - position() 方法

## 实例

获得 <p> 元素的当前位置：

```
$(".btn1").click(function(){
  x=$("p").position();
  $("#span1").text(x.left);
  $("#span2").text(x.top);
});
```

## 定义和用法

position() 方法返回匹配元素相对于父元素的位置（偏移）。

该方法返回的对象包含两个整型属性：top 和 left，以像素计。

此方法只对可见元素有效。

### 语法

```
$(_selector_).position()
```

# jQuery CSS 操作 - scrollLeft() 方法

## 实例

设置 <div> 元素中滚动条的水平偏移：

```
$(".btn1").click(function(){
  $("div").scrollLeft(100);
});
```

## 定义和用法

scrollLeft() 方法返回或设置匹配元素的滚动条的水平位置。

滚动条的水平位置指的是从其左侧滚动过的像素数。当滚动条位于最左侧时，位置是 0。

## 返回水平滚动条位置

返回第一个匹配元素的水平滚动条位置。

### 语法

```
$(_selector_).scrollLeft()
```

## 设置水平滚动条位置

设置所有匹配元素的水平滚动条位置。

### 语法

```
$(_selector_).scrollLeft(_position_)
```

| 参数 | 描述 |
| --- | --- |
| *position* | 可选。规定以像素计的新位置。 |

# jQuery CSS 操作 - scrollTop() 方法

## 实例

设置 <div> 元素中滚动条的垂直偏移：

```
$(".btn1").click(function(){
  $("div").scrollLeft(100);
});
```

## 定义和用法

scrollTop() 方法返回或设置匹配元素的滚动条的垂直位置。

scroll top offset 指的是滚动条相对于其顶部的偏移。

如果该方法未设置参数，则返回以像素计的相对滚动条顶部的偏移。

### 语法

```
$(_selector_).scrollTop(_offset_)
```

| 参数 | 描述 |
| --- | --- |
| offset | 可选。规定相对滚动条顶部的偏移，以像素计。 |

## 提示和注释

注释：该方法对于可见元素和不可见元素均有效。

注释：当用于获取值时，该方法只返回第一个匹配元素的 scroll top offset。

注释：当用于设置值时，该方法设置所有匹配元素的 scroll top offset。

## 亲自试一试 - 实例

获得当前的 scroll top offset

使用 scrollTop() 方法获得 scroll top offset。

# jQuery CSS 操作 - width() 方法

## 实例

设置 <p> 元素的宽度：

```
$(".btn1").click(function(){
  $("p").width(200);
});
```

## 定义和用法

width() 方法返回或设置匹配元素的宽度。

## 返回宽度

返回第一个匹配元素的宽度。

如果不为该方法设置参数，则返回以像素计的匹配元素的宽度。

### 语法

```
$(_selector_).width()
```

## 设置宽度

设置所有匹配元素的宽度。

### 语法

```
$(_selector_).width(_length_)
```

| 参数 | 描述 |
| --- | --- |
| *length* | 可选。规定元素的宽度。如果没有规定长度单位，则使用默认的 px 单位。 |

## 使用函数来设置宽度

使用函数来设置所有匹配元素的宽度。

## 语法

```
$(_selector_).width(_function(index,oldwidth)_)
```

| 参数 | 描述 |
| --- | --- |
| *function(index,oldwidth)* | 规定返回被选元素新宽度的函数。 `index` - 可选。接受选择器的 index 位置 `oldvalue` - 可选。接受选择器的当前值。 |

# 亲自试一试 - 实例

获得文档和窗口元素的宽度

使用 width() 方法来获得 document 和 window 元素的当前宽度。

使用 em 和 % 值来设置宽度

使用指定的长度单位来设置元素的宽度。

# jQuery 参考手册 - Ajax

## jQuery Ajax 操作函数

jQuery 库拥有完整的 Ajax 兼容套件。其中的函数和方法允许我们在不刷新浏览器的情况下从服务器加载数据。

| 函数 | 描述 |
|---|---|
| jQuery.ajax() | 执行异步 HTTP (Ajax) 请求。 |
| .ajaxComplete() | 当 Ajax 请求完成时注册要调用的处理程序。这是一个 Ajax 事件。 |
| .ajaxError() | 当 Ajax 请求完成且出现错误时注册要调用的处理程序。这是一个 Ajax 事件。 |
| .ajaxSend() | 在 Ajax 请求发送之前显示一条消息。 |
| jQuery.ajaxSetup() | 设置将来的 Ajax 请求的默认值。 |
| .ajaxStart() | 当首个 Ajax 请求完成开始时注册要调用的处理程序。这是一个 Ajax 事件。 |
| .ajaxStop() | 当所有 Ajax 请求完成时注册要调用的处理程序。这是一个 Ajax 事件。 |
| .ajaxSuccess() | 当 Ajax 请求成功完成时显示一条消息。 |
| jQuery.get() | 使用 HTTP GET 请求从服务器加载数据。 |
| jQuery.getJSON() | 使用 HTTP GET 请求从服务器加载 JSON 编码数据。 |
| jQuery.getScript() | 使用 HTTP GET 请求从服务器加载 JavaScript 文件，然后执行该文件。 |
| .load() | 从服务器加载数据，然后把返回到 HTML 放入匹配元素。 |
| jQuery.param() | 创建数组或对象的序列化表示，适合在 URL 查询字符串或 Ajax 请求中使用。 |
| jQuery.post() | 使用 HTTP POST 请求从服务器加载数据。 |
| .serialize() | 将表单内容序列化为字符串。 |
| .serializeArray() | 序列化表单元素，返回 JSON 数据结构数据。 |

## 参阅

教程：Ajax 教程

# jQuery ajax - ajax() 方法

## 实例

通过 AJAX 加载一段文本：

jQuery 代码：

```
$(document).ready(function(){
  $("#b01").click(function(){
  htmlobj=$.ajax({url:"/jquery/test1.txt",async:false});
  $("#myDiv").html(htmlobj.responseText);
  });
});
```

HTML 代码：

```
<div id="myDiv"><h2>Let AJAX change this text</h2></div>
<button id="b01" type="button">Change Content</button>
```

## 定义和用法

ajax() 方法通过 HTTP 请求加载远程数据。

该方法是 jQuery 底层 AJAX 实现。简单易用的高层实现见 $.get, $.post 等。$.ajax() 返回其创建的 XMLHttpRequest 对象。大多数情况下你无需直接操作该函数，除非你需要操作不常用的选项，以获得更多的灵活性。

最简单的情况下，$.ajax() 可以不带任何参数直接使用。

注意：所有的选项都可以通过 $.ajaxSetup() 函数来全局设置。

### 语法

```
jQuery.ajax([_settings_])
```

| 参数 | 描述 |
| --- | --- |
| *settings* | 可选。用于配置 Ajax 请求的键值对集合。可以通过 $.ajaxSetup() 设置任何选项的默认值。 |

### 参数

options

类型：Object

可选。AJAX 请求设置。所有选项都是可选的。

async

类型：Boolean

默认值: true。默认设置下，所有请求均为异步请求。如果需要发送同步请求，请将此选项设置为 false。

注意，同步请求将锁住浏览器，用户其它操作必须等待请求完成才可以执行。

beforeSend(XHR)

类型：Function

发送请求前可修改 XMLHttpRequest 对象的函数，如添加自定义 HTTP 头。

XMLHttpRequest 对象是唯一的参数。

这是一个 Ajax 事件。如果返回 false 可以取消本次 ajax 请求。

cache

类型：Boolean

默认值: true，dataType 为 script 和 jsonp 时默认为 false。设置为 false 将不缓存此页面。

jQuery 1.2 新功能。

complete(XHR, TS)

类型：Function

请求完成后回调函数 (请求成功或失败之后均调用)。

参数： XMLHttpRequest 对象和一个描述请求类型的字符串。

这是一个 Ajax 事件。

contentType

类型：String

默认值: "application/x-www-form-urlencoded"。发送信息至服务器时内容编码类型。

默认值适合大多数情况。如果你明确地传递了一个 content-type 给 $.ajax() 那么它必定会发送给服务器（即使没有数据要发送）。

context

类型：Object

这个对象用于设置 Ajax 相关回调函数的上下文。也就是说，让回调函数内 this 指向这个对象（如果不设定这个参数，那么 this 就指向调用本次 AJAX 请求时传递的 options 参数）。比如指定一个 DOM 元素作为 context 参数，这样就设置了 success 回调函数的上下文为这个 DOM 元素。

就像这样：

```
$.ajax({ url: "test.html", context: document.body, success: function(){
        $(this).addClass("done");
      }});
```

data

类型：String

发送到服务器的数据。将自动转换为请求字符串格式。GET 请求中将附加在 URL 后。查看 processData 选项说明以禁止此自动转换。必须为 Key/Value 格式。如果为数组，jQuery 将自动为不同值对应同一个名称。如 {foo:["bar1", "bar2"]} 转换为 '&foo=bar1&foo=bar2'。

dataFilter

类型：Function

给 Ajax 返回的原始数据的进行预处理的函数。提供 data 和 type 两个参数：data 是 Ajax 返回的原始数据，type 是调用 jQuery.ajax 时提供的 dataType 参数。函数返回的值将由 jQuery 进一步处理。

dataType

类型：String

预期服务器返回的数据类型。如果不指定，jQuery 将自动根据 HTTP 包 MIME 信息来智能判断，比如 XML MIME 类型就被识别为 XML。在 1.4 中，JSON 就会生成一个 JavaScript 对象，而 script 则会执行这个脚本。随后服务器端返回的数据会根据这个值解析后，传递给回调函数。可用值:

- "xml": 返回 XML 文档，可用 jQuery 处理。
- "html": 返回纯文本 HTML 信息；包含的 script 标签会在插入 dom 时执行。
- "script": 返回纯文本 JavaScript 代码。不会自动缓存结果。除非设置了 "cache" 参数。注意：在远程请求时(不在同一个域下)，所有 POST 请求都将转为 GET 请求。（因为将使用 DOM 的 script标签来加载）
- "json": 返回 JSON 数据 。
- "jsonp": JSONP 格式。使用 JSONP 形式调用函数时，如 "myurl?callback=?" jQuery 将

自动替换？为正确的函数名，以执行回调函数。
- "text": 返回纯文本字符串

error

类型：Function

默认值: 自动判断 (xml 或 html)。请求失败时调用此函数。

有以下三个参数：XMLHttpRequest 对象、错误信息、（可选）捕获的异常对象。

如果发生了错误，错误信息（第二个参数）除了得到 null 之外，还可能是 "timeout", "error", "notmodified" 和 "parsererror"。

这是一个 Ajax 事件。

global

类型：Boolean

是否触发全局 AJAX 事件。默认值: true。设置为 false 将不会触发全局 AJAX 事件，如 ajaxStart 或 ajaxStop 可用于控制不同的 Ajax 事件。

ifModified

类型：Boolean

仅在服务器数据改变时获取新数据。默认值: false。使用 HTTP 包 Last-Modified 头信息判断。在 jQuery 1.4 中，它也会检查服务器指定的 'etag' 来确定数据没有被修改过。

jsonp

类型：String

在一个 jsonp 请求中重写回调函数的名字。这个值用来替代在 "callback=?" 这种 GET 或 POST 请求中 URL 参数里的 "callback" 部分，比如 {jsonp:'onJsonPLoad'} 会导致将 "onJsonPLoad=?" 传给服务器。

jsonpCallback

类型：String

为 jsonp 请求指定一个回调函数名。这个值将用来取代 jQuery 自动生成的随机函数名。这主要用来让 jQuery 生成度独特的函数名，这样管理请求更容易，也能方便地提供回调函数和错误处理。你也可以在想让浏览器缓存 GET 请求的时候，指定这个回调函数名。

password

类型：String

用于响应 HTTP 访问认证请求的密码

processData

类型：Boolean

默认值: true。默认情况下，通过data选项传递进来的数据，如果是一个对象(技术上讲只要不是字符串)，都会处理转化成一个查询字符串，以配合默认内容类型 "application/x-www-form-urlencoded"。如果要发送 DOM 树信息或其它不希望转换的信息，请设置为 false。

scriptCharset

类型：String

只有当请求时 dataType 为 "jsonp" 或 "script"，并且 type 是 "GET" 才会用于强制修改 charset。通常只在本地和远程的内容编码不同时使用。

success

类型：Function

请求成功后的回调函数。

参数：由服务器返回，并根据 dataType 参数进行处理后的数据；描述状态的字符串。

这是一个 Ajax 事件。

traditional

类型：Boolean

如果你想要用传统的方式来序列化数据，那么就设置为 true。请参考工具分类下面的 jQuery.param 方法。

timeout

类型：Number

设置请求超时时间（毫秒）。此设置将覆盖全局设置。

type

类型：String

默认值: "GET")。请求方式 ("POST" 或 "GET")， 默认为 "GET"。注意：其它 HTTP 请求方法，如 PUT 和 DELETE 也可以使用，但仅部分浏览器支持。

url

类型：String

默认值: 当前页地址。发送请求的地址。

username

类型：String

用于响应 HTTP 访问认证请求的用户名。

xhr

类型：Function

需要返回一个 XMLHttpRequest 对象。默认在 IE 下是 ActiveXObject 而其他情况下是 XMLHttpRequest 。用于重写或者提供一个增强的 XMLHttpRequest 对象。这个参数在 jQuery 1.3 以前不可用。

# 回调函数

如果要处理 $.ajax() 得到的数据，则需要使用回调函数：beforeSend、error、dataFilter、success、complete。

## beforeSend

在发送请求之前调用，并且传入一个 XMLHttpRequest 作为参数。

## error

在请求出错时调用。传入 XMLHttpRequest 对象，描述错误类型的字符串以及一个异常对象（如果有的话）

## dataFilter

在请求成功之后调用。传入返回的数据以及 "dataType" 参数的值。并且必须返回新的数据（可能是处理过的）传递给 success 回调函数。

## success

当请求之后调用。传入返回后的数据，以及包含成功代码的字符串。

## complete

当请求完成之后调用这个函数，无论成功或失败。传入 XMLHttpRequest 对象，以及一个包含成功或错误代码的字符串。

# 数据类型

$.ajax() 函数依赖服务器提供的信息来处理返回的数据。如果服务器报告说返回的数据是 XML，那么返回的结果就可以用普通的 XML 方法或者 jQuery 的选择器来遍历。如果见得到其他类型，比如 HTML，则数据就以文本形式来对待。

通过 dataType 选项还可以指定其他不同数据处理方式。除了单纯的 XML，还可以指定 html、json、jsonp、script 或者 text。

其中，text 和 xml 类型返回的数据不会经过处理。数据仅仅简单的将 XMLHttpRequest 的 responseText 或 responseHTML 属性传递给 success 回调函数。

注意：我们必须确保网页服务器报告的 MIME 类型与我们选择的 dataType 所匹配。比如说，XML的话，服务器端就必须声明 text/xml 或者 application/xml 来获得一致的结果。

如果指定为 html 类型，任何内嵌的 JavaScript 都会在 HTML 作为一个字符串返回之前执行。类似地，指定 script 类型的话，也会先执行服务器端生成 JavaScript，然后再把脚本作为一个文本数据返回。

如果指定为 json 类型，则会把获取到的数据作为一个 JavaScript 对象来解析，并且把构建好的对象作为结果返回。为了实现这个目的，它首先尝试使用 JSON.parse()。如果浏览器不支持，则使用一个函数来构建。

JSON 数据是一种能很方便通过 JavaScript 解析的结构化数据。如果获取的数据文件存放在远程服务器上（域名不同，也就是跨域获取数据），则需要使用 jsonp 类型。使用这种类型的话，会创建一个查询字符串参数 callback=? ，这个参数会加在请求的 URL 后面。服务器端应当在 JSON 数据前加上回调函数名，以便完成一个有效的 JSONP 请求。如果要指定回调函数的参数名来取代默认的 callback，可以通过设置 $.ajax() 的 jsonp 参数。

注意：JSONP 是 JSON 格式的扩展。它要求一些服务器端的代码来检测并处理查询字符串参数。

如果指定了 script 或者 jsonp 类型，那么当从服务器接收到数据时，实际上是用了 <script> 标签而不是 XMLHttpRequest 对象。这种情况下，$.ajax() 不再返回一个 XMLHttpRequest 对象，并且也不会传递事件处理函数，比如 beforeSend。

# 发送数据到服务器

默认情况下，Ajax 请求使用 GET 方法。如果要使用 POST 方法，可以设定 type 参数值。这个选项也会影响 data 选项中的内容如何发送到服务器。

data 选项既可以包含一个查询字符串，比如 key1=value1&key2=value2 ，也可以是一个映射，比如 {key1: 'value1', key2: 'value2'} 。如果使用了后者的形式，则数据再发送器会被转换成查询字符串。这个处理过程也可以通过设置 processData 选项为 false 来回避。如果我们希

望发送一个 XML 对象给服务器时，这种处理可能并不合适。并且在这种情况下，我们也应当改变 contentType 选项的值，用其他合适的 MIME 类型来取代默认的 application/x-www-form-urlencoded 。

# 高级选项

global 选项用于阻止响应注册的回调函数，比如 .ajaxSend，或者 ajaxError，以及类似的方法。这在有些时候很有用，比如发送的请求非常频繁且简短的时候，就可以在 ajaxSend 里禁用这个。

如果服务器需要 HTTP 认证，可以使用用户名和密码可以通过 username 和 password 选项来设置。

Ajax 请求是限时的，所以错误警告被捕获并处理后，可以用来提升用户体验。请求超时这个参数通常就保留其默认值，要不就通过 jQuery.ajaxSetup 来全局设定，很少为特定的请求重新设置 timeout 选项。

默认情况下，请求总会被发出去，但浏览器有可能从它的缓存中调取数据。要禁止使用缓存的结果，可以设置 cache 参数为 false。如果希望判断数据自从上次请求后没有更改过就报告出错的话，可以设置 ifModified 为 true。

scriptCharset 允许给 <script> 标签的请求设定一个特定的字符集，用于 script 或者 jsonp 类似的数据。当脚本和页面字符集不同时，这特别好用。

Ajax 的第一个字母是 asynchronous 的开头字母，这意味着所有的操作都是并行的，完成的顺序没有前后关系。$.ajax() 的 async 参数总是设置成true，这标志着在请求开始后，其他代码依然能够执行。强烈不建议把这个选项设置成 false，这意味着所有的请求都不再是异步的了，这也会导致浏览器被锁死。

$.ajax 函数返回它创建的 XMLHttpRequest 对象。通常 jQuery 只在内部处理并创建这个对象，但用户也可以通过 xhr 选项来传递一个自己创建的 xhr 对象。返回的对象通常已经被丢弃了，但依然提供一个底层接口来观察和操控请求。比如说，调用对象上的 .abort() 可以在请求完成前挂起请求。

# jQuery ajax - ajaxComplete() 方法

## 实例

当 AJAX 请求正在进行时显示“正在加载”的指示：

```
$("#txt").ajaxStart(function(){
  $("#wait").css("display","block");
});
$("#txt").ajaxComplete(function(){
  $("#wait").css("display","none");
});
```

## 定义和用法

ajaxComplete() 方法在 AJAX 请求完成时执行函数。它是一个 Ajax 事件。

与 ajaxSuccess() 不同，通过 ajaxComplete() 方法规定的函数会在请求完成时运行，即使请求并未成功。

### 语法

```
.jQueryajaxComplete(_function(event,xhr,options)_)
```

| 参数 | 描述 |
| --- | --- |
| *function(event,xhr,options)* | 必需。规定当请求完成时运行的函数。额外的参数： *event* - 包含 event 对象 *xhr* - 包含 XMLHttpRequest 对象 *options* - 包含 AJAX 请求中使用的选项 |

### 详细说明

XMLHttpRequest 对象和设置作为参数传递给回调函数。

# jQuery ajax - ajaxError() 方法

## 实例

当 AJAX 请求失败时，触发提示框：

```
$("div").ajaxError(function(){
  alert("An error occurred!");
});
```

## 定义和用法

ajaxError() 方法在 AJAX 请求发生错误时执行函数。它是一个 Ajax 事件。

### 语法

```
.ajaxError(_function(event,xhr,options,exc)_)
```

| 参数 | 描述 |
| --- | --- |
| *function(event,xhr,options,exc)* | 必需。规定当请求失败时运行的函数。额外的参数：*event* - 包含 event 对象 *xhr* - 包含 XMLHttpRequest 对象 *options* - 包含 AJAX 请求中使用的选项 *exc* - 包含 JavaScript exception |

### 详细说明

XMLHttpRequest 对象和设置作为参数传递给回调函数。捕捉到的错误可作为最后一个参数传递：

```
function (event, XMLHttpRequest, ajaxOptions, thrownError) {
 // thrownError 只有当异常发生时才会被传递 this;
 }
```

## 亲自试一试 - 实例

使用 *xhr* 和 *options* 参数

如何使用 *options* 参数来获得更有用的错误消息。

# jQuery ajax - ajaxSend() 方法

## 实例

当 AJAX 请求即将发送时，改变 div 元素的内容：

```
$("div").ajaxSend(function(e,xhr,opt){
  $(this).html("Requesting " + opt.url);
});
```

## 定义和用法

ajaxSend() 方法在 AJAX 请求开始时执行函数。它是一个 Ajax 事件。

### 语法

```
.ajaxSend([_function(event,xhr,options)_])
```

| 参数 | 描述 |
| --- | --- |
| *function(event,xhr,options)* | 必需。规定当请求开始时执行函数。额外的参数： *event* - 包含 event 对象 *xhr* - 包含 XMLHttpRequest 对象 *options* - 包含 AJAX 请求中使用的选项 |

### 详细说明

XMLHttpRequest 对象和设置作为参数传递给回调函数。

# jQuery ajax - ajaxSetup() 方法

## 实例

为所有 AJAX 请求设置默认 URL 和 success 函数：

```
$("button").click(function(){
  $.ajaxSetup({url:"demo_ajax_load.txt",success:function(result){
    $("div").html(result);}});
  $.ajax();
});
```

## 定义和用法

jQuery.ajaxSetup() 方法设置全局 AJAX 默认选项。

### 语法

```
jQuery.ajaxSetup(_name:value_, _name:value_, ...)
```

### 示例

设置 AJAX 请求默认地址为 "/xmlhttp/"，禁止触发全局 AJAX 事件，用 POST 代替默认 GET 方法。其后的 AJAX 请求不再设置任何选项参数：

```
$.ajaxSetup({
  url: "/xmlhttp/",
  global: false,
  type: "POST"
});
$.ajax({ data: myData });
```

| 参数 | 描述 |
| --- | --- |
| *name:value* | 可选。使用名称/值对来规定 AJAX 请求的设置。 |

注释：参数见 '$.ajax' 说明。

# jQuery ajax - ajaxStart() 方法

## 实例

当 AJAX 请求开始时，显示“加载中”的指示：

```
$("div").ajaxStart(function(){
  $(this).html("<img src='demo_wait.gif' />");
});
```

## 定义和用法

ajaxStart() 方法在 AJAX 请求发送前执行函数。它是一个 Ajax 事件。

### 详细说明

无论在何时发送 Ajax 请求，jQuery 都会检查是否存在其他 Ajax 请求。如果不存在，则 jQuery 会触发该 ajaxStart 事件。在此时，由 .ajaxStart() 方法注册的任何函数都会被执行。

### 语法

```
.ajaxStart(_function()_)
```

| 参数 | 描述 |
| --- | --- |
| *function()* | 规定当 AJAX 请求开始时运行的函数。 |

### 示例

AJAX 请求开始时显示信息：

```
$("#loading").ajaxStart(function(){
  $(this).show();
});
```

# jQuery ajax - ajaxStop() 方法

## 实例

当所有 AJAX 请求完成时，触发一个提示框：

```
$("div").ajaxStop(function(){
  alert("所有 AJAX 请求已完成");
});
```

## 定义和用法

ajaxStop() 方法在 AJAX 请求结束时执行函数。它是一个 Ajax 事件。

### 详细说明

无论 Ajax 请求在何时完成，jQuery 都会检查是否存在其他 Ajax 请求。如果不存在，则 jQuery 会触发该 ajaxStop 事件。在此时，由 .ajaxStop() 方法注册的任何函数都会被执行。

### 语法

```
.ajaxStop(_function()_)
```

| 参数 | 描述 |
| --- | --- |
| *function()* | 规定当 AJAX 请求完成时运行的函数。 |

### 示例

AJAX 请求结束后隐藏信息：

```
$("#loading").ajaxStop(function(){
  $(this).hide();
});
```

# jQuery ajax - ajaxSuccess() 方法

## 实例

当 AJAX 请求成功完成时，触发提示框：

```
$("div").ajaxSuccess(function(){
  alert("AJAX 请求已成功完成");
});
```

## 定义和用法

ajaxSuccess() 方法在 AJAX 请求成功时执行函数。它是一个 Ajax 事件。

### 详细说明

XMLHttpRequest 对象和设置作为参数传递给回调函数。

无论 Ajax 请求在何时成功完成，jQuery 都会触发该 ajaxSuccess 事件。在此时，由 .ajaxSuccess() 方法注册的任何函数都会被执行。

### 语法

```
.ajaxSuccess(_function(event,xhr,options)_)
```

| 参数 | 描述 |
| --- | --- |
| *function(event,xhr,options)* | 必需。规定当请求成功时运行的函数。额外的参数： *event* - 包含 event 对象 *xhr* - 包含 XMLHttpRequest 对象 *options* - 包含 AJAX 请求中使用的选项 |

## 示例

AJAX 请求成功后显示消息：

```
$("#msg").ajaxSuccess(function(evt, request, settings){
  $(this).append("<p>请求成功!</p>");
});
```

# jQuery ajax - get() 方法

## 实例

使用 AJAX 的 GET 请求来改变 div 元素的文本：

```
$("button").click(function(){
  $.get("demo_ajax_load.txt", function(result){
    $("div").html(result);
  });
});
```

## 定义和用法

get() 方法通过远程 HTTP GET 请求载入信息。

这是一个简单的 GET 请求功能以取代复杂 $.ajax 。请求成功时可调用回调函数。如果需要在出错时执行函数，请使用 $.ajax。

### 语法

```
$(_selector_).get(_url_,_data_,_success(response,status,xhr)_,_dataType_)
```

| 参数 | 描述 |
| --- | --- |
| *url* | 必需。规定将请求发送的哪个 URL。 |
| *data* | 可选。规定连同请求发送到服务器的数据。 |
| *success(response,status,xhr)* | 可选。规定当请求成功时运行的函数。额外的参数：response - 包含来自请求的结果数据 status - 包含请求的状态 xhr - 包含 XMLHttpRequest 对象 |
| *dataType* | 可选。规定预计的服务器响应的数据类型。默认地，jQuery 将智能判断。可能的类型： "xml" "html" "text" "script" "json" "jsonp" |

### 详细说明

该函数是简写的 Ajax 函数，等价于：

```
$.ajax({
  url: _url_,
  data: _data_,
  success: _success_,
  dataType: _dataType_
});
```

根据响应的不同的 MIME 类型，传递给 success 回调函数的返回数据也有所不同，这些数据可以是 XML root 元素、文本字符串、JavaScript 文件或者 JSON 对象。也可向 success 回调函数传递响应的文本状态。

对于 jQuery 1.4，也可以向 success 回调函数传递 XMLHttpRequest 对象。

## 示例

请求 test.php 网页，忽略返回值：

```
$.get("test.php");
```

# 更多示例

## 例子 1

请求 test.php 网页，传送2个参数，忽略返回值：

```
$.get("test.php", { name: "John", time: "2pm" } );
```

## 例子 2

显示 test.php 返回值(HTML 或 XML，取决于返回值)：

```
$.get("test.php", function(data){
  alert("Data Loaded: " + data);
});
```

## 例子 3

显示 test.cgi 返回值(HTML 或 XML，取决于返回值)，添加一组请求参数：

```
$.get("test.cgi", { name: "John", time: "2pm" },
  function(data){
    alert("Data Loaded: " + data);
  });
```

# jQuery ajax - getJSON() 方法

## 实例

使用 AJAX 请求来获得 JSON 数据，并输出结果：

```
$("button").click(function(){
  $.getJSON("demo_ajax_json.js",function(result){
    $.each(result, function(i, field){
      $("div").append(field + " ");
    });
  });
});
```

## 定义和用法

通过 HTTP GET 请求载入 JSON 数据。

在 jQuery 1.2 中，您可以通过使用 JSONP 形式的回调函数来加载其他网域的 JSON 数据，如 "myurl?callback=?"。jQuery 将自动替换 ? 为正确的函数名，以执行回调函数。 注意：此行以后的代码将在这个回调函数执行前执行。

### 语法

```
jQuery.getJSON(_url_,_data_,_success(data,status,xhr)_)
```

| 参数 | 描述 |
| --- | --- |
| *url* | 必需。规定将请求发送的哪个 URL。 |
| *data* | 可选。规定连同请求发送到服务器的数据。 |
| *success(data,status,xhr)* | 可选。规定当请求成功时运行的函数。额外的参数： *response* - 包含来自请求的结果数据 *status* - 包含请求的状态 *xhr* - 包含 XMLHttpRequest 对象 |

### 详细说明

该函数是简写的 Ajax 函数，等价于：

```
$.ajax({
  url: _url_,
  data: _data_,
  success: _callback_,
  dataType: json
});
```

发送到服务器的数据可作为查询字符串附加到 URL 之后。如果 *data* 参数的值是对象（映射），那么在附加到 URL 之前将转换为字符串，并进行 URL 编码。

传递给 *callback* 的返回数据，可以是 JavaScript 对象，或以 JSON 结构定义的数组，并使用 $.parseJSON() 方法进行解析。

## 示例

从 test.js 载入 JSON 数据并显示 JSON 数据中一个 name 字段数据：

```
$.getJSON("test.js", function(json){
  alert("JSON Data: " + json.users[3].name);
});
```

# 更多示例

## 例子 1

从 Flickr JSONP API 载入 4 张最新的关于猫的图片：

HTML 代码：

```
<div id="images"></div>
```

jQuery 代码：

```
$.getJSON("http://api.flickr.com/services/feeds/photos_public.gne?
tags=cat&tagmode=any&format=json&jsoncallback=?", function(data){
  $.each(data.items, function(i,item){
    $("<img/>").attr("src", item.media.m).appendTo("#images");
    if ( i == 3 ) return false;
  });
});
```

## 例子 2

从 test.js 载入 JSON 数据，附加参数，显示 JSON 数据中一个 name 字段数据：

```
$.getJSON("test.js", { name: "John", time: "2pm" }, function(json){
  alert("JSON Data: " + json.users[3].name);
});
```

# jQuery ajax - getScript() 方法

## 实例

通过 AJAX 请求来获得并运行一个 JavaScript 文件：

```
$("button").click(function(){
  $.getScript("demo_ajax_script.js");
});
```

## 定义和用法

getScript() 方法通过 HTTP GET 请求载入并执行 JavaScript 文件。

### 语法

```
jQuery.getScript(_url_,_success(response,status)_)
```

| 参数 | 描述 |
| --- | --- |
| *url* | 将要请求的 URL 字符串。 |
| *success(response,status)* | 可选。规定请求成功后执行的回调函数。额外的参数：*response* - 包含来自请求的结果数据 *status* - 包含请求的状态（"success", "notmodified", "error", "timeout" 或 "parsererror"） |

### 详细说明

该函数是简写的 Ajax 函数，等价于：

```
$.ajax({
  url: _url_,
  dataType: "script",
  success: _success_
});
```

这里的回调函数会传入返回的 JavaScript 文件。这通常不怎么有用，因为那时脚本已经运行了。

载入的脚本在全局环境中执行，因此能够引用其他变量，并使用 jQuery 函数。

比如加载一个 test.js 文件，里边包含下面这段代码：

```
$(".result").html("<p>Lorem ipsum dolor sit amet.</p>");
```

通过引用该文件名，就可以载入并运行这段脚本：

```
$.getScript("ajax/test.js", function() {
  alert("Load was performed.");
});
```

注释：jQuery 1.2 版本之前，getScript 只能调用同域 JS 文件。 1.2中，您可以跨域调用 JavaScript 文件。注意：Safari 2 或更早的版本不能在全局作用域中同步执行脚本。如果通过 getScript 加入脚本，请加入延时函数。

# 更多实例

## 例子 1

加载并执行 test.js：

```
$.getScript("test.js");
```

## 例子 2

加载并执行 test.js ，成功后显示信息：

```
$.getScript("test.js", function(){
  alert("Script loaded and executed.");
});
```

## 例子 3

载入 jQuery 官方颜色动画插件 成功后绑定颜色变化动画：

HTML 代码：

```
<button id="go">Run</button>
<div class="block"></div>
```

jQuery 代码：

```
jQuery.getScript("http://dev.jquery.com/view/trunk/plugins/color/jquery.color.js",
 function(){
  $("#go").click(function(){
    $(".block").animate( { backgroundColor: 'pink' }, 1000)
      .animate( { backgroundColor: 'blue' }, 1000);
  });
});
```

# jQuery ajax - load() 方法

## 实例

使用 AJAX 请求来改变 div 元素的文本：

```
$("button").click(function(){
  $("div").load('demo_ajax_load.txt');
});
```

您可以在页面底部找到更多 TIY 实例

## 定义和用法

load() 方法通过 AJAX 请求从服务器加载数据，并把返回的数据放置到指定的元素中。

注释：还存在一个名为 load 的 jQuery 事件方法。调用哪个，取决于参数。

### 语法

```
load(_url_,_data_,_function(response,status,xhr)_)
```

| 参数 | 描述 |
| --- | --- |
| *url* | 规定要将请求发送到哪个 URL。 |
| *data* | 可选。规定连同请求发送到服务器的数据。 |
| *function(response,status,xhr)* | 可选。规定当请求完成时运行的函数。额外的参数：*response* - 包含来自请求的结果数据 *status* - 包含请求的状态（"success", "notmodified", "error", "timeout" 或 "parsererror"） *xhr* - 包含 XMLHttpRequest 对象 |

### 详细说明

该方法是最简单的从服务器获取数据的方法。它几乎与 $.get(url, data, success) 等价，不同的是它不是全局函数，并且它拥有隐式的回调函数。当侦测到成功的响应时（比如，当 textStatus 为 "success" 或 "notmodified" 时），.load() 将匹配元素的 HTML 内容设置为返回的数据。这意味着该方法的大多数使用会非常简单：

```
$("#result").load("ajax/test.html");
```

如果提供回调函数，则会在执行 post-processing 之后执行该函数：

```
$("#result").load("ajax/test.html", function() {
  alert("Load was performed.");
});
```

上面的两个例子中，如果当前文档不包含 "result" ID，则不会执行 .load() 方法。

如果提供的数据是对象，则使用 POST 方法；否则使用 GET 方法。

# 加载页面片段

.load() 方法，与 $.get() 不同，允许我们规定要插入的远程文档的某个部分。这一点是通过 url 参数的特殊语法实现的。如果该字符串中包含一个或多个空格，紧接第一个空格的字符串则是决定所加载内容的 jQuery 选择器。

我们可以修改上面的例子，这样就可以使用所获得文档的某部分：

```
$("#result").load("ajax/test.html #container");
```

如果执行该方法，则会取回 ajax/test.html 的内容，不过然后，jQuery 会解析被返回的文档，来查找带有容器 ID 的元素。该元素，连同其内容，会被插入带有结果 ID 的元素中，所取回文档的其余部分会被丢弃。

jQuery 使用浏览器的 .innerHTML 属性来解析被取回的文档，并把它插入当前文档。在此过程中，浏览器常会从文档中过滤掉元素，比如 <html>, <title> 或 <head> 元素。结果是，由 .load() 取回的元素可能与由浏览器直接取回的文档不完全相同。

注释：由于浏览器安全方面的限制，大多数 "Ajax" 请求遵守同源策略；请求无法从不同的域、子域或协议成功地取回数据。

# 更多实例

## 例子 **1**

加载 feeds.html 文件内容：

```
$("#feeds").load("feeds.html");
```

## 例子 **2**

与上面的实例类似，但是以 POST 形式发送附加参数并在成功时显示信息：

```
$("#feeds").load("feeds.php", {limit: 25}, function(){
  alert("The last 25 entries in the feed have been loaded");
});
```

## 例子 3

加载文章侧边栏导航部分至一个无序列表：

HTML 代码：

```
<b>jQuery Links:</b>
<ul id="links"></ul>
```

jQuery 代码：

```
$("#links").load("/Main_Page #p-Getting-Started li");
```

# 更多 TIY 实例

生成 AJAX 请求，并通过该请求发送数据

如何使用 data 参数通过 AJAX 请求来发送数据。（本例在 AJAX 教程中解释过。）

生成 AJAX 请求，并使用回调函数

如何使用 function 参数处理来自 AJAX 请求的数据结果。

生成带有错误的 AJAX 请求

如何使用 function 参数来处理 AJAX 请求中的错误（使用 XMLHttpRequest 参数）。

# jQuery ajax - param() 方法

## 实例

序列化一个 *key/value* 对象：

```
var params = { width:1900, height:1200 };
var str = jQuery.param(params);
$("#results").text(str);
```

结果:

```
width=1680&height=1050
```

### TIY 实例

输出序列化对象的结果：

```
$("button").click(function(){
  $("div").text($.param(personObj));
});
```

## 定义和用法

param() 方法创建数组或对象的序列化表示。

该序列化值可在进行 AJAX 请求时在 URL 查询字符串中使用。

### 语法

```
jQuery.param(_object_,_traditional_)
```

| 参数 | 描述 |
| --- | --- |
| *object* | 要进行序列化的数组或对象。 |
| *traditional* | 规定是否使用传统的方式浅层进行序列化（参数序列化）。 |

### 详细说明

param() 方法用于在内部将元素值转换为序列化的字符串表示。请参阅 .serialize() 了解更多信息。

对于 jQuery 1.3，如果传递的参数是一个函数，那么用 .param() 会得到这个函数的返回值，而不是把这个函数作为一个字符串来返回。

对于 jQuery 1.4，.param() 方法将会通过深度递归的方式序列化对象，以便符合现代化脚本语言的需求，比如 PHP、Ruby on Rails 等。你可以通过设置 jQuery.ajaxSettings.traditional = true; 来全局地禁用这个功能。

如果被传递的对象在数组中，则必须是以 .serializeArray() 的返回值为格式的对象数组：

```
[{name:"first",value:"Rick"},
{name:"last",value:"Astley"},
{name:"job",value:"Rock Star"}]
```

注意：因为有些框架在解析序列化数字的时候能力有限，所以当传递一些含有对象或嵌套数组的数组作为参数时，请务必小心！

在 jQuery 1.4 中，HTML5 的 input 元素也会被序列化。

# 更多实例

我们可以如下显示对象的查询字符串表示以及 URI 编码版本：

```
var myObject = {
  a: {
    one: 1,
    two: 2,
    three: 3
  },
  b: [1,2,3]
};
var recursiveEncoded = $.param(myObject);
var recursiveDecoded = decodeURIComponent($.param(myObject));

alert(recursiveEncoded);
alert(recursiveDecoded);
```

recursiveEncoded 和 recursiveDecoded 的值输出如下：

```
a%5Bone%5D=1&a%5Btwo%5D=2&a%5Bthree%5D=3&b%5B%5D=1&b%5B%5D=2&b%5B%5D=3
a[one]=1&a[two]=2&a[three]=3&b[]=1&b[]=2&b[]=3
```

可以将 traditional 参数设置为 true，来模拟 jQuery 1.4 之前版本中 $.param() 的行为：

```
var myObject = {
  a: {
    one: 1,
    two: 2,
    three: 3
  },
  b: [1,2,3]
};
var shallowEncoded = $.param(myObject, true);
var shallowDecoded = decodeURIComponent(shallowEncoded);

alert(shallowEncoded);
alert(shallowDecoded);
```

recursiveEncoded 和 recursiveDecoded 的值输出如下：

```
a=%5Bobject+Object%5D&b=1&b=2&b=3
a=[object+Object]&b=1&b=2&b=3
```

# jQuery ajax - post() 方法

## 实例

请求 test.php 网页，忽略返回值：

```
$.post("test.php");
```

### TIY 实例

通过 AJAX POST 请求改变 div 元素的文本：

```
$("input").keyup(function(){
  txt=$("input").val();
  $.post("demo_ajax_gethint.asp",{suggest:txt},function(result){
    $("span").html(result);
  });
});
```

## 定义和用法

post() 方法通过 HTTP POST 请求从服务器载入数据。

### 语法

```
jQuery.post(_url_,_data_,_success(data, textStatus, jqXHR)_,_dataType_)
```

| 参数 | 描述 |
| --- | --- |
| *url* | 必需。规定把请求发送到哪个 URL。 |
| *data* | 可选。映射或字符串值。规定连同请求发送到服务器的数据。 |
| *success(data, textStatus, jqXHR)* | 可选。请求成功时执行的回调函数。 |
| *dataType* | 可选。规定预期的服务器响应的数据类型。默认执行智能判断（xml、json、script 或 html）。 |

### 详细说明

该函数是简写的 Ajax 函数，等价于：

```
$.ajax({
  type: 'POST',
  url: _url_,
  data: _data_,
  success: _success_,
  dataType: _dataType_
});
```

根据响应的不同的 MIME 类型，传递给 success 回调函数的返回数据也有所不同，这些数据可以是 XML 根元素、文本字符串、JavaScript 文件或者 JSON 对象。也可向 success 回调函数传递响应的文本状态。

对于 jQuery 1.5，也可以向 success 回调函数传递 jqXHR 对象（jQuery 1.4 中传递的是 XMLHttpRequest 对象）。

大部分实现会规定一个 success 函数：

```
$.post("ajax/test.html", function(data) {
  $(".result").html(data);
});
```

本例读取被请求的 HTML 片段，并插入页面中。

通过 POST 读取的页面不被缓存，因此 jQuery.ajaxSetup() 中的 cache 和 ifModified 选项不会影响这些请求。

注释：由于浏览器安全方面的限制，大多数 "Ajax" 请求遵守同源策略；请求无法从不同的域、子域或协议成功地取回数据。

注释：如果由 jQuery.post() 发起的请求返回错误代码，那么不会有任何提示，除非脚本已调用了全局的 .ajaxError() 方法。或者对于 jQuery 1.5，jQuery.post() 返回的 jqXHR 对象的 .error() 方法也可以用于错误处理。

# jqXHR 对象

对于 jQuery 1.5，所有 jQuery 的 AJAX 方法返回的是 XMLHTTPRequest 对象的超集。由 $.post() 返回的 jQuery XHR 对象或 "jqXHR,"实现了约定的接口，赋予其所有的属性、方法，以及约定的行为。出于对由 $.ajax() 使用的回调函数名称便利性和一致性的考虑，它提供了 .error(), .success() 以及 .complete() 方法。这些方法使用请求终止时调用的函数参数，该函数接受与对应命名的 $.ajax() 回调函数相同的参数。

jQuery 1.5 中的约定接口同样允许 jQuery 的 Ajax 方法，包括 $.post()，来链接同一请求的多个 .success()、.complete() 以及 .error() 回调函数，甚至会在请求也许已经完成后分配这些回调函数。

```
// 请求生成后立即分配处理程序，请记住该请求针对 jqxhr 对象
    var jqxhr = $.post("example.php", function() {
      alert("success");
    })
    .success(function() { alert("second success"); })
    .error(function() { alert("error"); })
    .complete(function() { alert("complete"); });

    // 在这里执行其他任务

    // 为上面的请求设置另一个完成函数
    jqxhr.complete(function(){ alert("second complete"); });
```

# 更多实例

## 例子 1

请求 test.php 页面，并一起发送一些额外的数据（同时仍然忽略返回值）：

```
$.post("test.php", { name: "John", time: "2pm" } );
```

## 例子 2

向服务器传递数据数组（同时仍然忽略返回值）：

```
$.post("test.php", { 'choices[]': ["Jon", "Susan"] });
```

## 例子 3

使用 ajax 请求发送表单数据：

```
$.post("test.php", $("#testform").serialize());
```

## 例子 4

输出来自请求页面 test.php 的结果（HTML 或 XML，取决于所返回的内容）：

```
$.post("test.php", function(data){
   alert("Data Loaded: " + data);
 });
```

## 例子 5

向页面 test.php 发送数据，并输出结果（HTML 或 XML，取决于所返回的内容）：

```
$.post("test.php", { name: "John", time: "2pm" },
   function(data){
     alert("Data Loaded: " + data);
   });
```

## 例子 6

获得 test.php 页面的内容，并存储为 XMLHttpResponse 对象，并通过 process() 这个 JavaScript 函数进行处理：

```
$.post("test.php", { name: "John", time: "2pm" },
   function(data){
     process(data);
   }, "xml");
```

## 例子 7

获得 test.php 页面返回的 json 格式的内容：

```
$.post("test.php", { "func": "getNameAndTime" },
   function(data){
     alert(data.name); // John
     console.log(data.time); //  2pm
   }, "json");
```

# jQuery ajax - serialize() 方法

## 实例

输出序列化表单值的结果：

```
$("button").click(function(){
  $("div").text($("form").serialize());
});
```

## 定义和用法

serialize() 方法通过序列化表单值，创建 URL 编码文本字符串。

您可以选择一个或多个表单元素（比如 input 及/或 文本框），或者 form 元素本身。

序列化的值可在生成 AJAX 请求时用于 URL 查询字符串中。

### 语法

```
$(_selector_).serialize()
```

### 详细说明

.serialize() 方法创建以标准 URL 编码表示的文本字符串。它的操作对象是代表表单元素集合的 jQuery 对象。

表单元素有几种类型：

```
<form>
  <div><input type="text" name="a" value="1" id="a" /></div>
  <div><input type="text" name="b" value="2" id="b" /></div>
  <div><input type="hidden" name="c" value="3" id="c" /></div>
  <div>
    <textarea name="d" rows="8" cols="40">4</textarea>
  </div>
  <div><select name="e">
    <option value="5" selected="selected">5</option>
    <option value="6">6</option>
    <option value="7">7</option>
  </select></div>
  <div>
    <input type="checkbox" name="f" value="8" id="f" />
  </div>
  <div>
    <input type="submit" name="g" value="Submit" id="g" />
  </div>
</form>
```

.serialize() 方法可以操作已选取个别表单元素的 jQuery 对象，比如 <input>, <textarea> 以及 <select>。不过，选择 <form> 标签本身进行序列化一般更容易些：

```
$('form').submit(function() {
  alert($(this).serialize());
  return false;
});
```

输出标准的查询字符串：

```
a=1&b=2&c=3&d=4&e=5
```

注释：只会将"成功的控件"序列化为字符串。如果不使用按钮来提交表单，则不对提交按钮的值序列化。如果要表单元素的值包含到序列字符串中，元素必须使用 name 属性。

# jQuery ajax - serializeArray() 方法

## 实例

输出以数组形式序列化表单值的结果：

```
$("button").click(function(){
  x=$("form").serializeArray();
  $.each(x, function(i, field){
    $("#results").append(field.name + ":" + field.value + " ");
  });
});
```

## 定义和用法

serializeArray() 方法通过序列化表单值来创建对象数组（名称和值）。

您可以选择一个或多个表单元素（比如 input 及/或 textarea），或者 form 元素本身。

### 语法

```
$(_selector_).serializeArray()
```

### 详细说明

serializeArray() 方法序列化表单元素（类似 .serialize() 方法），返回 JSON 数据结构数据。

注意：此方法返回的是 JSON 对象而非 JSON 字符串。需要使用插件或者第三方库进行字符串化操作。

返回的 JSON 对象是由一个对象数组组成的，其中每个对象包含一个或两个名值对 —— name 参数和 value 参数（如果 value 不为空的话）。举例来说：

```
[
  {name: 'firstname', value: 'Hello'},
  {name: 'lastname', value: 'World'},
  {name: 'alias'}, // 值为空
]
```

.serializeArray() 方法使用了 W3C 关于 successful controls（有效控件） 的标准来检测哪些元素应当包括在内。特别说明，元素不能被禁用（禁用的元素不会被包括在内），并且元素应当有含有 name 属性。提交按钮的值也不会被序列化。文件选择元素的数据也不会被序列

化。

该方法可以对已选择单独表单元素的对象进行操作，比如 <input>, <textarea>, 和 <select>。不过，更方便的方法是，直接选择 <form> 标签自身来进行序列化操作。

```
$("form").submit(function() {
  console.log($(this).serializeArray());
  return false;
});
```

上面的代码产生下面的数据结构（假设浏览器支持 console.log）：

```
[
  {
    name: a
    value: 1
  },
  {
    name: b
    value: 2
  },
  {
    name: c
    value: 3
  },
  {
    name: d
    value: 4
  },
  {
    name: e
    value: 5
  }
]
```

# 示例

取得表单内容并插入到网页中：

HTML 代码：

```
<p id="results"><b>Results:</b> </p>
<form>
  <select name="single">
    <option>Single</option>
    <option>Single2</option>
  </select>
  <select name="multiple" multiple="multiple">
    <option selected="selected">Multiple</option>
    <option>Multiple2</option>
    <option selected="selected">Multiple3</option>
  </select><br/>
  <input type="checkbox" name="check" value="check1"/> check1
  <input type="checkbox" name="check" value="check2" checked="checked"/> check2
  <input type="radio" name="radio" value="radio1" checked="checked"/> radio1
  <input type="radio" name="radio" value="radio2"/> radio2
</form>
```

jQuery 代码：

```
var fields = $("select, :radio").serializeArray();
jQuery.each( fields, function(i, field){
  $("#results").append(field.value + " ");
});
```

# jQuery 参考手册 - 遍历

## jQuery 遍历函数

jQuery 遍历函数包括了用于筛选、查找和串联元素的方法。

| 函数 | 描述 |
| --- | --- |
| .add() | 将元素添加到匹配元素的集合中。 |
| .andSelf() | 把堆栈中之前的元素集添加到当前集合中。 |
| .children() | 获得匹配元素集合中每个元素的所有子元素。 |
| .closest() | 从元素本身开始，逐级向上级元素匹配，并返回最先匹配的祖先元素。 |
| .contents() | 获得匹配元素集合中每个元素的子元素，包括文本和注释节点。 |
| .each() | 对 jQuery 对象进行迭代，为每个匹配元素执行函数。 |
| .end() | 结束当前链中最近的一次筛选操作，并将匹配元素集合返回到前一次的状态。 |
| .eq() | 将匹配元素集合缩减为位于指定索引的新元素。 |
| .filter() | 将匹配元素集合缩减为匹配选择器或匹配函数返回值的新元素。 |
| .find() | 获得当前匹配元素集合中每个元素的后代，由选择器进行筛选。 |
| .first() | 将匹配元素集合缩减为集合中的第一个元素。 |
| .has() | 将匹配元素集合缩减为包含特定元素的后代的集合。 |
| .is() | 根据选择器检查当前匹配元素集合，如果存在至少一个匹配元素，则返回 true。 |
| .last() | 将匹配元素集合缩减为集合中的最后一个元素。 |
| .map() | 把当前匹配集合中的每个元素传递给函数，产生包含返回值的新 jQuery 对象。 |
| .next() | 获得匹配元素集合中每个元素紧邻的同辈元素。 |
| .nextAll() | 获得匹配元素集合中每个元素之后的所有同辈元素，由选择器进行筛选（可选）。 |
| .nextUntil() | 获得每个元素之后所有的同辈元素，直到遇到匹配选择器的元素为止。 |
| .not() | 从匹配元素集合中删除元素。 |
| .offsetParent() | 获得用于定位的第一个父元素。 |
| .parent() | 获得当前匹配元素集合中每个元素的父元素，由选择器筛选（可选）。 |
|  | 获得当前匹配元素集合中每个元素的祖先元素，由选择器筛选（可 |

| | |
|---|---|
| | 选）。 |
| .parentsUntil() | 获得当前匹配元素集合中每个元素的祖先元素，直到遇到匹配选择器的元素为止。 |
| .prev() | 获得匹配元素集合中每个元素紧邻的前一个同辈元素，由选择器筛选（可选）。 |
| .prevAll() | 获得匹配元素集合中每个元素之前的所有同辈元素，由选择器进行筛选（可选）。 |
| .prevUntil() | 获得每个元素之前所有的同辈元素，直到遇到匹配选择器的元素为止。 |
| .siblings() | 获得匹配元素集合中所有元素的同辈元素，由选择器筛选（可选）。 |
| .slice() | 将匹配元素集合缩减为指定范围的子集。 |

# jQuery 遍历 - add() 方法

## 实例

找到所有 div 并添加边框。然后将所有段落添加到该 jQuery 对象，并把它们的背景设置为黄色：

```
$("div").css("border", "2px solid red")
        .add("p")
        .css("background", "yellow");
```

## 定义和用法

add() 方法将元素添加到匹配元素的集合中。

### 语法 1

```
.add(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 字符串值，表示查找供添加到匹配元素集合的元素的选择器表达式。 |

### 语法 2

```
.add(_elements_)
```

| 参数 | 描述 |
| --- | --- |
| *elements* | 添加到匹配元素集合的一个或多个元素。 |

### 语法 3

```
.add(_html_)
```

| 参数 | 描述 |
| --- | --- |
| *html* | 添加到匹配元素集合的 HTML 片段。 |

## 语法 4

```
.add(_jQueryObject_)
```

| 参数 | 描述 |
| --- | --- |
| *jQueryObject* | 添加到匹配元素集合的已有 jQuery 对象。 |

## 语法 5

```
.add(_selector_, _context_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 字符串值，表示查找供添加到匹配元素集合的元素的选择器表达式。 |
| *context* | 选择器开始进行匹配的位置。 |

## 详细说明

暂无。

# jQuery 遍历 - andSelf() 方法

## 实例

找到所有 div，以及其中的所有段落，并为它们添加两个类名。请注意，由于未使用 .andSelf()，div 没有黄色背景色。

```
$("div").find("p").andSelf().addClass("border");
$("div").find("p").addClass("background");
```

## 定义和用法

add() 方法把堆栈中之前的元素集添加到当前集合。

### 语法

```
.andSelf()
```

### 详细说明

请思考这个拥有简单列表的页面：

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li class="third-item">list item 3</li>
   <li>list item 4</li>
   <li>list item 5</li>
</ul>
```

以下代码的结果是项目 3,4,5 拥有红色背景：

```
$("li.third-item").nextAll().andSelf()
  .css("background-color", "red");
```

首先，初始的选择器会定位项目 3，初始化的堆栈存有仅包含该项目的集合。调用 .nextAll() 会将项目 4, 5 的集合推入堆栈。最后，调用 .andSelf() 会合并这两个集合，所创建的 jQuery 对象指向按照文档顺序的所有三个项目：{[<li.third-item>,<li>,<li> ]}。

# jQuery 遍历 - children() 方法

## 实例

找到类名为 "selected" 的所有 div 的子元素，并将其设置为蓝色：

```
$("div").children(".selected").css("color", "blue");
```

## 定义和用法

add() 方法返回匹配元素集合中每个元素的子元素，添加可选参数可通过选择器进行过滤。

### 语法

```
.children(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 字符串值，包含匹配元素的选择器表达式。 |

### 详细说明

如果给定表示 DOM 元素集合的 jQuery 对象，.children() 方法允许我们检索 DOM 树中的这些元素，并用匹配元素构造新的 jQuery 对象。.find() 和 .children() 方法类似，不过后者只沿着 DOM 树向下遍历单一层级。

请注意，与大多数 jQuery 方法一样，.children() 不返回文本节点；如果需要获得包含文本和注释节点在内的所有子节点，请使用 .contents()。

该方法接受一个选择器表达式作为可选参数，与我们传递到 $() 的参数的类型是相同的。如果应用该选择器，将测试元素是否匹配该表达式，以此筛选这些元素。

请思考这个带有基础的嵌套列表的页面：

```
<ul class="level-1">
  <li class="item-i">I</li>
  <li class="item-ii">II
    <ul class="level-2">
      <li class="item-a">A</li>
      <li class="item-b">B
        <ul class="level-3">
          <li class="item-1">1</li>
          <li class="item-2">2</li>
          <li class="item-3">3</li>
        </ul>
      </li>
      <li class="item-c">C</li>
    </ul>
  </li>
  <li class="item-iii">III</li>
</ul>
```

如果我们从 level-2 列表开始，我们可以找到它的子元素：

```
$('ul.level-2').children().css('background-color', 'red');
```

这行代码的结果是，项目 A, B, C 得到红色背景。由于我们没有应用选择器表达式，返回的 jQuery 对象包含了所有子元素。如果应用一个选择器的话，那么只会包括匹配的项目。

# jQuery 遍历 - closest() 方法

## 实例

本例演示如何通过 closest() 完成事件委托。当被最接近的列表元素或其子后代元素被点击时，会切换黄色背景：

```
$( document ).bind("click", function( e ) {
    $( e.target ).closest("li").toggleClass("hilight");
  });
```

## 定义和用法

closest() 方法获得匹配选择器的第一个祖先元素，从当前元素开始沿 DOM 树向上。

### 语法

```
.closest(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 字符串值，包含匹配元素的选择器表达式。 |

### 详细说明

如果给定表示 DOM 元素集合的 jQuery 对象，.closest() 方法允许我们检索 DOM 树中的这些元素以及它们的祖先元素，并用匹配元素构造新的 jQuery 对象。.parents() 和 .closest() 方法类似，它们都沿 DOM 树向上遍历。两者之间的差异尽管微妙，却很重要：

| .closest() | .parents() |
| --- | --- |
| 从当前元素开始 | 从父元素开始 |
| 沿 DOM 树向上遍历，直到找到已应用选择器的一个匹配为止。 | 沿 DOM 树向上遍历，直到文档的根元素为止，将每个祖先元素添加到一个临时的集合；如果应用了选择器，则会基于该选择器对这个集合进行筛选。 |
| 返回包含零个或一个元素的 jQuery 对象 | 返回包含零个、一个或多个元素的 jQuery 对象 |

请看下面的 HTML 片段：

```
<ul id="one" class="level-1">
  <li class="item-i">I</li>
  <li id="ii" class="item-ii">II
  <ul class="level-2">
    <li class="item-a">A</li>
    <li class="item-b">B
      <ul class="level-3">
        <li class="item-1">1</li>
        <li class="item-2">2</li>
        <li class="item-3">3</li>
      </ul>
    </li>
    <li class="item-c">C</li>
  </ul>
  </li>
  <li class="item-iii">III</li>
</ul>
```

## 例子 1

假设我们执行一个从项目 A 开始的对 <ul> 元素的搜索：

```
$('li.item-a').closest('ul').css('background-color', 'red');
```

这会改变 level-2 <ul> 的颜色，这是因为当向上遍历 DOM 树时会第一个遇到该元素。

## 例子 2

假设我们搜索的是 <li> 元素：

```
$('li.item-a').closest('li').css('background-color', 'red');
```

这会改变列表项目 A 的颜色。在向上遍历 DOM 树之前，.closest() 方法会从 li 元素本身开始搜索，直到选择器匹配项目 A 为止。

## 例子 3

我们可以传递 DOM 元素作为 context，在其中搜索最接近的元素。

```
var listItemII = document.getElementById('ii');
$('li.item-a').closest('ul', listItemII).css('background-color', 'red');
$('li.item-a').closest('#one', listItemII).css('background-color', 'green');
```

以上代码会改变 level-2 <ul> 的颜色，因为它既是列表项 A 的第一个 <ul> 祖先，同时也是列表项 II 的后代。它不会改变 level-1 <ul> 的颜色，因为它不是 list item II 的后代。

# jQuery 遍历 - contents() 方法

## 实例

找到段落中的所有文本节点，并用粗体标签包装它们。

```
$("p").contents().filter(function(){ return this.nodeType != 1; }).wrap("<b/>");
```

## 定义和用法

contents() 方法获得匹配元素集合中每个元素的子节点，包括文本和注释节点。

### 语法

```
.contents()
```

### 详细说明

如果给定表示 DOM 元素集合的 jQuery 对象，.contents() 方法允许我们检索 DOM 树中的这些元素的直接子节点，并用匹配元素构造新的 jQuery 对象。.contents() 和 .children() 方法类似，不同的是前者在结果 jQuery 对象中包含了文本节点以及 HTML 元素。

.contents() 方法也可以用于获得 iframe 的内容文档，前提是该 iframe 与主页面在同一个域。

请思考下面这个带有一些文本节点的 <div>，每个节点被两个折行元素 (<br />) 分隔：

```
<div class="container">
  Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed
  do eiusmod tempor incididunt ut labore et dolore magna aliqua.
  <br /><br />
  Ut enim ad minim veniam, quis nostrud exercitation ullamco
  laboris nisi ut aliquip ex ea commodo consequat.
  <br /> <br />
  Duis aute irure dolor in reprehenderit in voluptate velit
  esse cillum dolore eu fugiat nulla pariatur.
</div>
```

我们可以使用 .contents() 方法来把文本块转换为形式良好的段落：

```
$('.container').contents().filter(function() {
  return this.nodeType == 3;
})
  .wrap('<p></p>')
.end()
.filter('br')
  .remove();
```

这段代码首先会接收 <div class="container"> 的内容，然后滤过其文本节点，将文本节点封装入段落标签中。这是通过测试元素的 .nodeType 属性实现的。该属性存有指示节点类型的数字代码；文本节点使用代码 3。内容会被再次过滤，这次针对 <br /> 元素，这些元素会被移除。

# jQuery 遍历 - each() 方法

## 实例

输出每个 li 元素的文本：

```
$("button").click(function(){
  $("li").each(function(){
    alert($(this).text())
  });
});
```

## 定义和用法

each() 方法规定为每个匹配元素规定运行的函数。

提示：返回 false 可用于及早停止循环。

### 语法

```
$(_selector_).each(_function(index,element)_)
```

| 参数 | 描述 |
| --- | --- |
| *function(index,element)* | 必需。为每个匹配元素规定运行的函数。 *index* - 选择器的 index 位置 *element* - 当前的元素（也可使用 "this" 选择器） |

# jQuery 遍历 - end() 方法

## 实例

选择所有段落，找到这些段落中的 span 元素，然后将它们恢复为段落，并把段落设置为两像素的红色边框：

```
$("p").find("span").end().css("border", "2px red solid");
```

## 定义和用法

end() 方法结束当前链条中的最近的筛选操作，并将匹配元素集还原为之前的状态。

### 语法

```
.end()
```

### 详细说明

大多数 jQuery 的遍历方法会操作一个 jQuery 对象实例，并生成一个匹配不同 DOM 元素集的新对象。当发生这种情况时，应该会把新的元素集推入维持在对象中的堆栈内。每次成功的筛选方法调用都会把新元素推入堆栈中。如果我们需要老的元素集，可以使用 end() 从堆栈中弹出新集合。

假设页面中有一对很短的列表：

```
<ul class="first">
   <li class="foo">list item 1</li>
   <li>list item 2</li>
   <li class="bar">list item 3</li>
</ul>
<ul class="second">
   <li class="foo">list item 1</li>
   <li>list item 2</li>
   <li class="bar">list item 3</li>
</ul>
```

## 例子 1

主要是在利用 jQuery 的链条属性（命令链）时，jQuery 会比较有用。如果不使用命令链，我们一般是通过变量名来调用之前的对象，这样我们就不需要操作堆栈了。不过通过 end()，我们可以把所有方法调用串联在一起：

```
$('ul.first').find('.foo').css('background-color', 'red')
  .end().find('.bar').css('background-color', 'green');
```

这条命令链检索第一个列表中类名为 foo 的项目，并把它们的背景设置为红色。end() 会将对象还原为调用 find() 之前的状态，所以第二个 find() 查找的是 <ul class="first"> 内的 '.bar'，而不是在列表的 <li class="foo"> 中查找，并将匹配元素的背景设置为绿色。最后的结果是第一个列表中的项目 1 和项目 3 被设置了带颜色的背景，而第二个列表中的项目没有任何变化。

## 例子 2

这条长长的 jQuery 链可以可视化为结构化的代码块，筛选方法打开嵌套代码块，而 end() 方法用来关闭代码块：

```
$('ul.first').find('.foo')
  .css('background-color', 'red')
.end().find('.bar')
  .css('background-color', 'green')
.end();
```

最后这个 end() 不是必需的，因为我们随后会丢弃这个 jQuery 对象。不过，如果按照这种形式编写代码，end() 就能提供视觉上的对称，以及规整程序的感觉，至少对于开发者来说更易阅读，当然代价则是由于进行了额外的调用，会有一点点性能损失。

# jQuery 遍历 - eq() 方法

## 实例

通过为 index 为 2 的 div 添加适当的类，将其变为蓝色：

```
$("body").find("div").eq(2).addClass("blue");
```

## 定义和用法

eq() 方法将匹配元素集缩减值指定 index 上的一个。

### 语法

```
.eq(_index_)
```

| 参数 | 描述 |
| --- | --- |
| *index* | 整数，指示元素的位置（最小为 0）。如果是负数，则从集合中的最后一个元素往回计数。 |

### 详细说明

如果给定表示 DOM 元素集合的 jQuery 对象，.eq() 方法会用集合中的一个元素构造一个新的 jQuery 对象。所使用的 index 参数标示集合中元素的位置。

请看下面这个简单的列表：

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

## 例子 1

我们可以把该方法应用到这个列表项目集：

```
$('li').eq(2).css('background-color', 'red');
```

这个调用的结果是为项目 3 设置了红色背景。请注意，index 是基于零的，并且是在 jQuery 对象中引用元素的位置，而不是在 DOM 树中。

## 例子 2

如果提供负数，则指示从集合结尾开始的位置，而不是从开头开始。例如：

```
$('li').eq(-2).css('background-color', 'red');
```

这次，项目 4 的背景变为红色，这是因为它是集合结尾开始的第二个。

## 例子 3

如果无法根据指定的 index 参数找到元素，则该方法构造带有空集的 jQuery 对象，length 属性为 0。

```
$('li').eq(5).css('background-color', 'red');
```

这里，没有列表项会变为红色，这是因为 .eq(5) 指示的第六个列表项。

# jQuery 遍历 - filter() 方法

## 实例

改变所有 div 的颜色，然后向类名为 "middle" 的类添加边框：

```
$("div").css("background", "#c8ebcc")
  .filter(".middle")
  .css("border-color", "red");
```

## 定义和用法

filter() 方法将匹配元素集合缩减为匹配指定选择器的元素。

### 语法

```
.filter(_selector_)
```

| 参数 | 描述 |
|---|---|
| *selector* | 字符串值，包含供匹配当前元素集合的选择器表达式。 |

### 详细说明

如果给定表示 DOM 元素集合的 jQuery 对象，.filter() 方法会用匹配元素的子集构造一个新的 jQuery 对象。所使用的选择器会测试每个元素；所有匹配该选择器的元素都会包含在结果中。

请思考下面这个拥有简单列表的页面：

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
  <li>list item 6</li>
</ul>
```

我们可以向这个列表项集合应用该方法：

```
$('li').filter(':even').css('background-color', 'red');
```

此调用的结果是将项目 1, 3, 5 的背景设置为红色，这是因为它们都匹配选择器（回忆一下，:even 和 :odd 均使用基于 0 的 index）。

# 使用过滤函数

使用该方法的第二个形式是，通过函数而不是选择器来筛选元素。对于每个元素，如果该函数返回 true，则元素会被包含在已筛选集合中；否则，会排除这个元素。

请看下面这段稍显复杂的 HTML 片段：

```
<ul>
  <li><strong>list</strong> item 1 - one strong tag</li>
  <li><strong>list</strong> item <strong>2</strong>
   - two <span>strong tags</span></li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
  <li>list item 6</li>
</ul>
```

我们可以选取这些列表项，然后基于其内容来筛选它们：

```
$('li').filter(function(index) {
  return $('strong', this).length == 1;
}).css('background-color', 'red');
```

# jQuery 遍历 - find() 方法

## 实例

搜索所有段落中的后代 span 元素，并将其颜色设置为红色：

```
$("p").find("span").css('color','red');
```

## 定义和用法

find() 方法获得当前元素集合中每个元素的后代，通过选择器、jQuery 对象或元素来筛选。

### 语法

```
.find(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 字符串值，包含供匹配当前元素集合的选择器表达式。 |

### 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.find() 方法允许我们在 DOM 树中搜索这些元素的后代，并用匹配元素来构造一个新的 jQuery 对象。.find() 与 .children() 方法类似，不同的是后者仅沿着 DOM 树向下遍历单一层级。

.find() 方法第一个明显特征是，其接受的选择器表达式与我们向 $() 函数传递的表达式的类型相同。将通过测试这些元素是否匹配该表达式来对元素进行过滤。

请思考下面这个简单的嵌套列表：

```
<ul class="level-1">
  <li class="item-i">I</li>
  <li class="item-ii">II
    <ul class="level-2">
      <li class="item-a">A</li>
      <li class="item-b">B
        <ul class="level-3">
          <li class="item-1">1</li>
          <li class="item-2">2</li>
          <li class="item-3">3</li>
        </ul>
      </li>
      <li class="item-c">C</li>
    </ul>
  </li>
  <li class="item-iii">III</li>
</ul>
```

我们将从列表 II 开始来查找其中的列表项：

```
$('li.item-ii').find('li').css('background-color', 'red');
```

这次调研的结果是，项目 A、B、1、2、3 以及 C 均被设置了红色背景。即使项目 II 匹配选择器表达式，它也不会被包含在结果中；只会对后代进行匹配。

与其他的树遍历方法不同，选择器表达式对于 .find() 是必需的参数。如果我们需要实现对所有后代元素的取回，可以传递通配选择器 '*'。

选择器 context 是由 .find() 方法实现的；因此，$('li.item-ii').find('li') 等价于 $('li', 'li.item-ii')。

对于 jQuery 1.6，我们还可以使用给定的 jQuery 集合或元素来进行筛选。还是上面的嵌套列表，我们首先这样写：

```
var $allListElements = $('li');
```

然后将这个 jQuery 对象传递给 find 方法：

```
$('li.item-ii').find( $allListElements );
```

上面的代码会返回一个 jQuery 集合，其中包含属于列表 II 后代的列表元素。

类似地，也可以传递一个元素：

```
var item1 = $('li.item-1')[0];
$('li.item-ii').find( item1 ).css('background-color', 'red');
```

这次调用的结果是项目 1 被设置为红色背景。

# jQuery 遍历 - first() 方法

## 实例

高亮显示段落中的第一个 span ：

```
$("p span").first().addClass('highlight');
```

## 定义和用法

first() 将匹配元素集合缩减为集合中的第一个元素。

### 语法

```
.first()
```

### 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.first() 方法会用第一个匹配元素构造一个新的 jQuery 对象。

请思考下面这个带有简单列表的页面：

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

我们可以对这个列表项集合应用该方法：

```
$('li').first().css('background-color', 'red');
```

这次调用的结果是，第一个项目被设置为红色背景。

# jQuery 遍历 - has() 方法

## 实例

检测某个元素是否在另一个元素中：

```
$("ul").append("<li>" + ($("ul").has("li").length ? "Yes" : "No") + "</li>");
$("ul").has("li").addClass("full");
```

## 定义和用法

has() 将匹配元素集合缩减为拥有匹配指定选择器或 DOM 元素的后代的子集。

### 语法

```
.has(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 字符串值，包含匹配元素的选择器表达式。 |

### 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.has() 方法用匹配元素的子集来构造一个新的 jQuery 对象。所使用的选择器用于检测匹配元素的后代；如果任何后代元素匹配该选择器，该元素将被包含在结果中。

请思考下面这个带有嵌套列表的页面：

```
<ul>
  <li>list item 1</li>
  <li>list item 2
    <ul>
      <li>list item 2-a</li>
      <li>list item 2-b</li>
    </ul>
  </li>
  <li>list item 3</li>
  <li>list item 4</li>
</ul>
```

我们可以对列表项集合应用该方法，就像这样：

```
$('li').has('ul').css('background-color', 'red');
```

该调用的结果是，项目 2 的背景被设置为红色，这是因为该项目是后代中唯一拥有 <ul> 的 <li>。

# jQuery 遍历 - is() 方法

## 实例

返回 false，因为 input 元素的父元素是 p 元素：

```
var isFormParent = $("input[type='checkbox']").parent().is("form");
$("div").text("isFormParent = " + isFormParent);
```

## 定义和用法

is() 根据选择器、元素或 jQuery 对象来检测匹配元素集合，如果这些元素中至少有一个元素匹配给定的参数，则返回 true。

### 语法

```
.is(_selector_)
```

| 参数 | 描述 |
|---|---|
| *selector* | 字符串值，包含匹配元素的选择器表达式。 |

### 详细说明

与其他筛选方法不同，.is() 不创建新的 jQuery 对象。相反，它允许我们在不修改 jQuery 对象内容的情况下对其进行检测。这在 callback 内部通常比较有用，比如事件处理程序。

假设我们有一个列表，其中两个项目包含子元素：

```
<ul>
  <li>list <strong>item 1</strong></li>
  <li><span>list item 2</span></li>
  <li>list item 3</li>
</ul>
```

您可以向 <ul> 元素添加 click 处理程序，然后把代码限制为只有当列表项本身，而非子元素，被点击时才进行触发：

```
$("ul").click(function(event) {
  var $target = $(event.target);
  if ( $target.is("li") ) {
    $target.css("background-color", "red");
  }
});
```

现在，当用户点击的是第一个列表项中的单词 "list" 或第三个列表项中的任何单词时，被点击的列表项会被设置为红色背景。不过，当用户点击第一个列表项中的 item 1 或第二个列表项中的任何单词时，都不会有任何变化，这是因为这上面的情况中，事件的目标分别是 <strong> 是 <span>。

请您注意，对于带有位置性选择器的选择器表达式字符串，比如 :first、:gt() 或者 :even，位置性筛选是针对传递到 .is() 的 jQuery 对象进行的，而非针对包含文档。所以对于上面的 HTML 来说，诸如 $("li:first").is("li:last") 的表达式返回 true，但是 $("li:first-child").is("li:last-child") 返回 false。

# 使用函数

该方法的第二种用法是，对基于函数而非选择器的相关元素的表达式进行求值。对于每个元素来说，如果该函数返回 true，则 .is() 也返回 true。例如，下面是稍微复杂的 HTML 片段：

```
<ul>
  <li><strong>list</strong> item 1 - one strong tag</li>
  <li><strong>list</strong> item <strong>2</strong> -
    two <span>strong tags</span></li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

您可以向每个 <li> 添加 click 处理程序，以计算在被点击的 <li> 内部 <strong> 元素的数目：

```
$("li").click(function() {
  var $li = $(this),
    isWithTwo = $li.is(function() {
      return $('strong', this).length === 2;
    });
  if ( isWithTwo ) {
    $li.css("background-color", "green");
  } else {
    $li.css("background-color", "red");
  }
});
```

# jQuery 遍历 - last() 方法

## 实例

高亮显示段落中的最后一个 span ：

```
$("p span").last().addClass('highlight');
```

## 定义和用法

last() 将匹配元素集合缩减为集合中的最后一个元素。

### 语法

```
.last()
```

### 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.last() 方法会用最后一个匹配元素构造一个新的 jQuery 对象。

请思考下面这个带有简单列表的页面：

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

我们可以对这个列表项集合应用该方法：

```
$('li').last().css('background-color', 'red');
```

这次调用的结果是，最后一个项目被设置为红色背景。

# jQuery 遍历 - map() 方法

## 实例

构建表单中所有值的列表：

```
$("p").append( $("input").map(function(){
  return $(this).val();
}).get().join(", ") );
```

## 定义和用法

map() 把每个元素通过函数传递到当前匹配集合中，生成包含返回值的新的 jQuery 对象。

### 语法

```
.map(_callback(index,domElement)_)
```

| 参数 | 描述 |
|---|---|
| *callback(index,domElement)* | 对当前集合中的每个元素调用的函数对象。 |

### 详细说明

由于返回值是 jQuery 封装的数组，使用 get() 来处理返回的对象以得到基础的数组。

.map() 方法对于获得或设置元素集的值特别有用。请思考下面这个带有一系列复选框的表单：

```
<form method="post" action="">
  <fieldset>
    <div>
      <label for="two">2</label>
      <input type="checkbox" value="2" id="two" name="number[]">
    </div>
    <div>
      <label for="four">4</label>
      <input type="checkbox" value="4" id="four" name="number[]">
    </div>
    <div>
      <label for="six">6</label>
      <input type="checkbox" value="6" id="six" name="number[]">
    </div>
    <div>
      <label for="eight">8</label>
      <input type="checkbox" value="8" id="eight" name="number[]">
    </div>
  </fieldset>
</form>
```

我们能够获得复选框 ID 组成的逗号分隔的列表：

```
$(':checkbox').map(function() {
  return this.id;
}).get().join(',');
```

本次调用的结果是字符串："two,four,six,eight"。

在 callback 函数内部，this 引用每次迭代的当前 DOM 元素。该函数可返回单独的数据项，或者是要被插入结果集中的数据项的数组。如果返回的是数组，数组内的元素会被插入集合中。如果函数返回 null 或 undefined，则不会插入任何元素。

# jQuery 遍历 - next() 方法

## 实例

查找每个段落的下一个同胞元素，仅选中类名为 "selected" 的段落：

```
$("p").next(".selected").css("background", "yellow");
```

## 定义和用法

next() 获得匹配元素集合中每个元素紧邻的同胞元素。如果提供选择器，则取回匹配该选择器的下一个同胞元素。

### 语法

```
.next(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 字符串值，包含用于匹配元素的选择器表达式。 |

### 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.next() 方法允许我们搜索 DOM 树中的元素紧跟的同胞元素，并用匹配元素构造新的 jQuery 对象。

该方法接受可选的选择器表达式，类型与我传递到 $() 函数中的相同。如果紧跟的同胞元素匹配该选择器，则会留在新构造的 jQuery 对象中；否则会将之排除。

请思考下面这个带有简单列表的页面：

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li class="third-item">list item 3</li>
   <li>list item 4</li>
   <li>list item 5</li>
</ul>
```

如果我们从项目三开始，则能够找到其后出现的元素：

```
$('li.third-item').next().css('background-color', 'red');
```

这次调用的结果是，项目 4 被设置为红色背景。由于我们没有应用选择器表达式，紧跟的这个元素很明确地被包括为对象的一部分。如果我们已经应用了选择器，在包含它之前会检测是否匹配。

# jQuery 遍历 - nextAll() 方法

## 实例

查找第一个 div 之后的所有类名，并为他们添加类名：

```
$("div:first").nextAll().addClass("after");
```

## 定义和用法

nextAll() 获得匹配元素集合中每个元素的所有跟随的同胞元素，由选择器筛选是可选的。

### 语法

```
.nextAll(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 字符串值，包含用于匹配元素的选择器表达式。 |

### 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.nextAll() 方法允许我们搜索 DOM 树中的元素跟随的同胞元素，并用匹配元素构造新的 jQuery 对象。

该方法接受可选的选择器表达式，类型与我传递到 $() 函数中的相同。如果应用选择器，则将通过检测元素是否匹配来对它们进行筛选。

请思考下面这个带有简单列表的页面：

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li class="third-item">list item 3</li>
   <li>list item 4</li>
   <li>list item 5</li>
</ul>
```

如果我们从项目三开始，那么我们能够找到其后出现的元素：

```
$('li.third-item').nextAll().css('background-color', 'red');
```

这次调用的结果是，项目 4 和 5 被设置为红色背景。由于我们没有应用选择器表达式，紧跟的这个元素很明确地被包括为对象的一部分。如果我们已经应用了选择器，在包含它之前会检测是否匹配。

# jQuery 遍历 - nextUntil() 方法

## 实例

查找跟随 <dt id="term-2"> 的同胞元素，直到下一个 <dt>，然后将它们设置为红色背景色。同时，找到跟随 <dt id="term-1"> 的 <dd> 同胞元素，直到 <dt id="term-3">，并为它们设置蓝色文本颜色。

```
$("#term-2").nextUntil("dt").css("background-color", "red");
var term3 = document.getElementById("term-3");
$("#term-1").nextUntil(term3, "dd").css("color", "blue");
```

## 定义和用法

nextUntil() 获得每个元素所有跟随的同胞元素，但不包括被选择器、DOM 节点或已传递的 jQuery 对象匹配的元素。

### 语法 1

```
.nextUntil(_selector_,_filter_)
```

| 参数 | 描述 |
|---|---|
| *selector* | 字符串值，包含指示在何处停止匹配跟随的同胞元素的选择器表达式。 |
| *filter* | 字符串值，包含用于匹配元素的选择器表达式。 |

### 语法 2

```
.nextUntil(_element_,_filter_)
```

| 参数 | 描述 |
|---|---|
| *element* | 指示在何处停止匹配跟随的同胞元素的 DOM 节点或 jQuery 对象。 |
| *filter* | 字符串值，包含用于匹配元素的选择器表达式。 |

### 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.nextUntil() 方法允许我们搜索 DOM 树中的元素跟随的同胞元素，当遇到被该方法的参数所匹配的元素时会停止搜索。返回的新 jQuery 对象包含所有跟随的同胞元素，但不包含被参数匹配的元素。

如果选择器不匹配或未规定选择器，则会选取所有跟随的同胞；如果不提供供筛选的选择器，则该方法选取的元素与 .nextAll() 方法相同。

对于 jQuery 1.6，DOM 节点或 jQuery 对象，而不是选择器，可传递到 .nextUntil() 方法。

该方法接受可选的选择器表达式作为其第二参数。如果指定该参数，则将通过检测元素是否匹配该选择器来筛选它们。

# jQuery 遍历 - not() 方法

## 实例

从包含所有段落的集合中删除 id 为 "selected" 的段落：

```
$("p").not("#selected")
```

## 定义和用法

not() 从匹配元素集合中删除元素。

### 语法 1

```
.not(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 字符串值，包含用于匹配元素的选择器表达式。 |

### 语法 2

```
.not(_element_)
```

| 参数 | 描述 |
| --- | --- |
| *element* | 一个或多个需要从匹配集中删除的 DOM 元素。 |

### 语法 3

```
.not(_function(index)_)
```

| 参数 | 描述 |
| --- | --- |
| *function(index)* | 用于检测集合中每个元素的函数。this 是当前 DOM 元素。 |

## 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.not() 方法会用匹配元素的子集构造一个新的 jQuery 对象。所应用的选择器会检测每个元素；不匹配该选择器的元素会被包含在结果中。

请思考下面这个带有简单列表的页面：

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

我们可以向列表项集应用该方法：

```
$('li').not(':even').css('background-color', 'red');
```

这次调用的结果是将项目 2 和 4 设置为红色背景，这是因为它们不匹配选择器（回忆一下，:even 和 :odd 均使用基于 0 的 index）。

## 移除具体的元素

.not() 方法的第二个版本允许我们从匹配集中删除元素，假设我们之前已经通过其他手段找到了这些元素。例如，设想一个列表已经将 id 应用到其中一个项目中：

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li id="notli">list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

我们可以使用原生的 JavaScript 函数 getElementById() 读取第三个列表项，然后把它从 jQuery 对象中删除：

```
$('li').not(document.getElementById('notli')).css('background-color', 'red');
```

这条语句改变项目 1、2、3 和 5 的背景色。我们可以用更简单的 jQuery 表达式来完成同样的事情，但是这项技术在比方说其他库提供对纯 DOM 节点的引用时会很有用。

对于 jQuery 1.4，.not() 方法能够采用函数作为其参数，与 .filter() 方法相同。其函数返回 true 的元素会被排除在过滤集之外；所有其他元素将被包含其中。

# jQuery 遍历 - offsetParent() 方法

## 实例

设置类名为 item-a 的 li 元素的最近定位父元素的背景色：

```
$('li.item-a').offsetParent().css('background-color', 'red');
```

## 定义和用法

offsetParent() 获得被定位的最近祖先元素。

### 语法

```
.offsetParent()
```

### 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.offsetParent() 方法允许我们搜索 DOM 树中元素的祖先，并构造一个由最近的定位祖先元素包围的 jQuery 对象。定位元素指的是，元素的 CSS position 属性设置为 relative、absolute 或 fixed。在为表演动画计算偏移或在页面上放置对象时，该信息会很有用处。

请思考带有基本嵌套列表的页面，其中带有定位元素：

```
<ul class="level-1">
  <li class="item-i">I</li>
  <li class="item-ii" style="position: relative;">II
    <ul class="level-2">
      <li class="item-a">A</li>
      <li class="item-b">B
        <ul class="level-3">
          <li class="item-1">1</li>
          <li class="item-2">2</li>
          <li class="item-3">3</li>
        </ul>
      </li>
      <li class="item-c">C</li>
    </ul>
  </li>
  <li class="item-iii">III</li>
</ul>
```

如果我们从项目 A 开始，我们可以找到其定位祖先元素：

```
$('li.item-a').offsetParent().css('background-color', 'red');
```

这会改变被定位的项目 II 的背景色。

# jQuery 遍历 - parent() 方法

## 实例

查找每个段落的带有 "selected" 类的父元素：

```
$("p").parent(".selected")
```

## 定义和用法

parent() 获得当前匹配元素集合中每个元素的父元素，使用选择器进行筛选是可选的。

```
.parent(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 字符串值，包含用于匹配元素的选择器表达式。 |

### 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.parent() 方法允许我们在 DOM 树中搜索这些元素的父元素，并用匹配元素构造一个新的 jQuery 对象。.parents() 和 .parent() 方法类似，不同的是后者沿 DOM 树向上遍历单一层级。

该方法接受可选的选择器表达式，与我们向 $() 函数中传递的参数类型相同。如果应用这个选择器，则将通过检测元素是否匹配该选择器对元素进行筛选。

请思考这个带有基本的嵌套列表的页面：

```
<ul class="level-1">
  <li class="item-i">I</li>
  <li class="item-ii">II
    <ul class="level-2">
      <li class="item-a">A</li>
      <li class="item-b">B
        <ul class="level-3">
          <li class="item-1">1</li>
          <li class="item-2">2</li>
          <li class="item-3">3</li>
        </ul>
      </li>
      <li class="item-c">C</li>
    </ul>
  </li>
  <li class="item-iii">III</li>
</ul>
```

如果我们从项目 A 开始，则可找到其父元素：

```
$('li.item-a').parent().css('background-color', 'red');
```

此次调用的结果是，为 level-2 列表设置红色背景。由于我们未应用选择器表达式，父元素很自然地成为了对象的一部分。如果已应用选择器，则会在包含元素之前，检测元素是否匹配选择器。

# jQuery 遍历 - parents() 方法

## 实例

查找每个 b 元素的所有父元素：

```
$("b").parents()
```

## 定义和用法

parents() 获得当前匹配元素集合中每个元素的祖先元素，使用选择器进行筛选是可选的。

```
.parents(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 字符串值，包含用于匹配元素的选择器表达式。 |

### 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.parents() 方法允许我们在 DOM 树中搜索这些元素的祖先元素，并用从最近的父元素向上的顺序排列的匹配元素构造一个新的 jQuery 对象。元素是按照从最近的父元素向外的顺序被返回的。.parents() 和 .parent() 方法类似，不同的是后者沿 DOM 树向上遍历单一层级。

该方法接受可选的选择器表达式，与我们向 $() 函数中传递的参数类型相同。如果应用这个选择器，则将通过检测元素是否匹配该选择器对元素进行筛选。

请思考这个带有基本的嵌套列表的页面：

```
<ul class="level-1">
  <li class="item-i">I</li>
  <li class="item-ii">II
    <ul class="level-2">
      <li class="item-a">A</li>
      <li class="item-b">B
        <ul class="level-3">
          <li class="item-1">1</li>
          <li class="item-2">2</li>
          <li class="item-3">3</li>
        </ul>
      </li>
      <li class="item-c">C</li>
    </ul>
  </li>
  <li class="item-iii">III</li>
</ul>
```

如果我们从项目 A 开始，则可找到其祖先元素：

```
$('li.item-a').parents().css('background-color', 'red');
```

此次调用的结果是，level-2 列表、项目 II 以及 level-1 列表等元素（沿 DOM 树一路向上直到 <html>）设置红色背景。由于我们未应用选择器表达式，父元素很自然地成为了对象的一部分。如果已应用选择器，则会在包含元素之前，检测元素是否匹配选择器。由于我们未应用选择器表达式，所有祖先元素都是返回的 jQuery 对象的组成部分。如果已应用选择器，则只会包含其中的匹配项目。

# jQuery 遍历 - parentsUntil() 方法

## 实例

查找 <li class="item-a"> 的祖先元素，直到 <ul class="level-1">，并将它们设置为红色背景。同时，找到 <li class="item-2"> 的所有类名为 "yes" 的祖先元素，直到 <ul class="level-1">，然户为它们设置蓝色边框：

```
$("li.item-a").parentsUntil(".level-1")
  .css("background-color", "red");

$("li.item-2").parentsUntil( $("ul.level-1"), ".yes" )
  .css("border", "3px solid blue");
```

## 定义和用法

parentsUntil() 获得当前匹配元素集合中每个元素的祖先元素，直到（但不包括）被选择器、DOM 节点或 jQuery 对象匹配的元素。

### 语法 1

```
.parentsUntil(_selector_,_filter_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 可选。字符串值，规定在何处停止对祖先元素进行匹配的选择器表达式。 |
| *filter* | 可选。字符串值，包含用于匹配元素的选择器表达式。 |

### 语法 2

```
.parentsUntil(_element_,_filter_)
```

| 参数 | 描述 |
| --- | --- |
| *element* | 可选。DOM 节点或 jQuery 对象，指示在何处停止对祖先元素的匹配。 |
| *filter* | 可选。字符串值，包含用于匹配元素的选择器表达式。 |

## 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.parentsUntil() 方法允许我们在 DOM 树中搜索这些元素的祖先元素，直到遇到被选择器（传递到方法中的参数）匹配的元素为止。返回的 jQuery 对象包含所有祖先元素，但不包括由 .parentsUntil() 方法规定的选择器匹配的那个元素。

如果不匹配或未应用选择器，则将选区所有祖先元素；在这种情况下，该方法选取的元素与未提供选择器时的 .parents() 相同。

对于 jQuery 1.6，DOM 节点或 jQuery 对象，而不是选择器，可用作 .parentsUntil() 方法的第一个参数。

该方法接受可选的选择器表达式作为其第二参数。如果应用这个参数，则将通过检测元素是否匹配该选择器对元素进行筛选。

# jQuery 遍历 - prev() 方法

## 实例

检索每个段落，找到类名为 "selected" 的前一个同胞元素：

```
$("p").prev(".selected")
```

## 定义和用法

prev() 获得匹配元素集合中每个元素紧邻的前一个同胞元素，通过选择器进行筛选是可选的。

```
.prev(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 字符串值，包含用于匹配元素的选择器表达式。 |

### 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.prev() 方法允许我们在 DOM 树中搜索这些元素的前一个同胞元素，并用匹配元素构造一个新的 jQuery 对象。

该方法接受可选的选择器表达式，与我们向 $() 函数中传递的参数类型相同。如果应用这个选择器，则将通过检测元素是否匹配该选择器对元素进行筛选。

请思考这个带有基本的嵌套列表的页面：

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li class="third-item">list item 3</li>
   <li>list item 4</li>
   <li>list item 5</li>
</ul>
```

如果我们从第三个项目开始，则可找到该元素之间的紧邻元素：

```
$('li.third-item').prev().css('background-color', 'red');
```

此处调用的结果是将项目 2 设置为红色背景。由于我们未应用选择器表达式，前一个元素很自然地成为了对象的一部分。如果已应用选择器，则会在包含元素之前，检测元素是否匹配选择器。

# jQuery 遍历 - prevAll() 方法

## 实例

定位最后一个 div 之前的所有 div，并为它们添加类：

```
$("div:last").prevAll().addClass("before");
```

## 定义和用法

prevAll() 获得当前匹配元素集合中每个元素的前面的同胞元素，使用选择器进行筛选是可选的。

### 语法

```
.prevAll(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 字符串值，包含用于匹配元素的选择器表达式。 |

### 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.prevAll() 方法允许我们在 DOM 树中搜索这些元素前面的同胞元素，并用匹配元素构造一个新的 jQuery 对象。

该方法接受可选的选择器表达式，与我们向 $() 函数中传递的参数类型相同。如果应用这个选择器，则将通过检测元素是否匹配该选择器对元素进行筛选。

请思考这个带有基本的嵌套列表的页面：

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li class="third-item">list item 3</li>
   <li>list item 4</li>
   <li>list item 5</li>
</ul>
```

如果我们从第三个项目开始，则可找到该元素之间的同胞元素：

```
$('li.third-item').prevAll().css('background-color', 'red');
```

此处调用的结果是将项目 2 和项目 1 设置为红色背景。由于我们未应用选择器表达式，这些前面的元素很自然地成为了对象的一部分。如果已应用选择器，则会在包含元素之前，检测这些元素是否匹配选择器。

# jQuery 遍历 - prevUntil() 方法

## 实例

查找 <dt id="term-2"> 之前的同胞元素，直到前一个 <dt>，并将它们设置为红色。同时，查找 <dt id="term-3"> 前面的 <dd> 同胞，直到 <dt id="term-1">，并把它们设置为蓝色文本：

```
$("#term-2").prevUntil("dt").css("background-color", "red");
var term1 = document.getElementById('term-1');
$("#term-3").prevUntil(term1, "dd").css("color", "green");
```

## 定义和用法

prevUntil() 方法获得当前匹配元素集合中每个元素的前面的同胞元素，但不包括被选择器、DOM 节点或 jQuery 对象匹配的元素。

### 语法 1

```
.prevUntil(_selector_, _filter_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 可选。字符串值，包含指示在何处停止匹配前方同胞元素的选择器表达式。 |
| *filter* | 可选。字符串值，包含用于匹配元素的选择器表达式。 |

### 语法 2

```
.prevUntil(_element_, _filter_)
```

| 参数 | 描述 |
| --- | --- |
| *element* | 可选。指示在何处停止匹配前方同胞元素的 DOM 节点或 jQuery 对象。 |
| *filter* | 可选。字符串值，包含用于匹配元素的选择器表达式。 |

### 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.prevUntil() 方法允许我们在 DOM 树中搜索这些元素前面的同胞元素，直到遇到被选择器（传递到方法中的参数）匹配的元素为止。返回的 jQuery 新对象包含所有前面的同胞元素，但不包括由 .prevUntil() 方法规定的选择器匹配的那个元素；所返回元素的顺序是从最近的同胞元素到最远的那个。

如果不匹配或未应用选择器，则将选区所有前面的同胞元素；在这种情况下，该方法选取的元素与未提供选择器时的 .prevAll() 相同。

对于 jQuery 1.6，DOM 节点或 jQuery 对象，而不是选择器，可用作 .prevUntil() 方法的第一个参数。

该方法接受可选的选择器表达式作为其第二参数。如果应用这个参数，则将通过检测元素是否匹配该选择器对元素进行筛选。

# jQuery 遍历 - siblings() 方法

## 实例

查找每个 p 元素的所有类名为 "selected" 的所有同胞元素：

```
$("p").siblings(".selected")
```

## 定义和用法

siblings() 获得匹配集合中每个元素的同胞，通过选择器进行筛选是可选的。

### 语法

```
.siblings(_selector_)
```

| 参数 | 描述 |
| --- | --- |
| *selector* | 字符串值，包含用于匹配元素的选择器表达式。 |

### 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.siblings() 方法允许我们在 DOM 树中搜索这些元素的同胞元素，并用匹配元素构造一个新的 jQuery 对象。

该方法接受可选的选择器表达式，与我们向 $() 函数中传递的参数类型相同。如果应用这个选择器，则将通过检测元素是否匹配该选择器对元素进行筛选。

请思考这个带有基本的嵌套列表的页面：

```
<ul>
   <li>list item 1</li>
   <li>list item 2</li>
   <li class="third-item">list item 3</li>
   <li>list item 4</li>
   <li>list item 5</li>
</ul>
```

如果我们从第三个项目开始，则可找到该元素的同胞元素：

```
$('li.third-item').siblings().css('background-color', 'red');
```

此处调用的结果是将项目 1、2、4 和 5 的背景设置为红色。 设置为红色背景。由于我们未应用选择器表达式，所有同胞元素很自然地成为了对象的一部分。如果已应用选择器，则只会包含四个列表中的匹配的项目。

原始元素不包含在同胞元素中，当我们打算找到 DOM 树的特定层级上的所有元素时，记住一点很重要。

# jQuery 遍历 - slice() 方法

## 实例

选中所有段落，然后将所选内容缩减为只包含第一和第二个段落：

```
$("p").slice(0, 2).wrapInner("");
```

## 定义和用法

slice() 把匹配元素集合缩减为指定的指数范围的子集。

### 语法

```
.slice(_selector_,_end_)
```

| 参数 | 描述 |
|---|---|
| *selector* | 基于 0 的整数值，指示开始选取元素的位置。如果是负数，则指示从集合末端开始的偏移量。 |
| *end* | 基于 0 的整数值，指示结束选取元素的位置。如果是负数，则指示从集合末端开始的偏移量。如果省略，则选取范围会在集合末端结束。 |

### 详细说明

如果给定一个表示 DOM 元素集合的 jQuery 对象，.slice() 方法用匹配元素的子集构造一个新的 jQuery 对象。已应用的 index 参数集合中其中一个元素的位置；如果省略 end 参数，则 index 之后的所有的所有元素都会包含在结果中。

请思考这个带有简单列表的页面：

```
<ul>
  <li>list item 1</li>
  <li>list item 2</li>
  <li>list item 3</li>
  <li>list item 4</li>
  <li>list item 5</li>
</ul>
```

我们可以向列表项集合应用该方法：

```
$('li').slice(2).css('background-color', 'red');
```

此次调用的结果是项目 3、4 和 5 的背景被设置为红色。请注意，已应用的 index 参数基于零，引用的是 jQuery 对象中元素的位置，而非 DOM 树中的。

end 参数允许我们更进一步地限制选取范围。比如：

```
$('li').slice(2, 4).css('background-color', 'red');
```

现在，只有项目 3 和 4 会被选取。再次说明，index 是基于零的；范围会延伸到（但不包含）指定的 index。

## 负的指数

jQuery 的 .slice() 方法模仿了 JavaScript 数组对象的 .slice() 方法。它所模仿的特性之一是向 start 或 end 参数传递负数的能力。如果提供负数，则指示的是从集合结尾开始的一个位置，而非从开头。例如：

```
$('li').slice(-2, -1).css('background-color', 'red');
```

这次，只有列表项 4 会变红，这是因为该项目是介于从结尾计数的二 (-2) 与从结尾计数的一 (-1) 的之间的范围中的唯一项目。

# jQuery 参考手册 - 数据

## jQuery 数据操作函数

这些方法允许我们将指定的 DOM 元素与任意数据相关联。

| 函数 | 描述 |
| --- | --- |
| .clearQueue() | 从队列中删除所有未运行的项目。 |
| .data() | 存储与匹配元素相关的任意数据。 |
| jQuery.data() | 存储与指定元素相关的任意数据。 |
| .dequeue() | 从队列最前端移除一个队列函数，并执行它。 |
| jQuery.dequeue() | 从队列最前端移除一个队列函数，并执行它。 |
| jQuery.hasData() | 存储与匹配元素相关的任意数据。 |
| .queue() | 显示或操作匹配元素所执行函数的队列。 |
| jQuery.queue() | 显示或操作匹配元素所执行函数的队列。 |
| .removeData() | 移除之前存放的数据。 |
| jQuery.removeData() | 移除之前存放的数据。 |

## 参阅

参考手册：jQuery 队列控制

# jQuery 遍历 - clearQueue() 方法

## 实例

清空队列：

```
$("div").clearQueue();
```

## 定义和用法

clearQueue() 方法从序列中删除仍未运行的所有项目。

### 语法

```
.clearQueue(_queueName_)
```

| 参数 | 描述 |
| --- | --- |
| *queueName* | 可选。字符串值，包含序列的名称。默认是 fx, 标准的效果序列。 |

### 详细说明

当调用 .clearQueue() 方法时，序列中未被执行的所有函数都会被从序列中删除。如果不使用参数，则 .clearQueue() 从 fx （标准效果序列）中删除剩余的函数。在这种方式中，它类似于 .stop(true)。不过，.stop() 方法只用于动画，而 .clearQueue() 也可用于删除通过 .queue() 方法添加到通用 jQuery 序列的任何函数。

# jQuery 数据 - data() 方法

## 实例

向元素附加数据，然后取回该数据：

```
$("#btn1").click(function(){
  $("div").data("greeting", "Hello World");
});
$("#btn2").click(function(){
  alert($("div").data("greeting"));
});
```

## 定义和用法

data() 方法向被选元素附加数据，或者从被选元素获取数据。

## 从元素返回数据

从被选元素中返回附加的数据。

### 语法

```
$(_selector_).data(_name_)
```

| 参数 | 描述 |
| --- | --- |
| *name* | 可选。规定要取回的数据的名称。如果没有规定名称，则该方法将以对象的形式从元素中返回所有存储的数据。 |

## 向元素附加数据

向被选元素附加数据。

### 语法

```
$(_selector_).data(_name_,_value_)
```

| 参数 | 描述 |
| --- | --- |
| *name* | 必需。规定要设置的数据的名称。 |
| *value* | 必需。规定要设置的数据的值。 |

# 使用对象向元素附加数据

使用带有名称/值对的对象向被选元素添加数据。

## 语法

```
$(_selector_).data(_object_)
```

| 参数 | 描述 |
| --- | --- |
| *object* | 必需。规定包含名称/值对的对象。 |

# jQuery 数据 - jQuery.data() 方法

## 实例

向元素附加数据，然后取回该数据：

```
$("#btn1").click(function(){
  $("div").data("greeting", "Hello World");
});
$("#btn2").click(function(){
  alert($("div").data("greeting"));
});
```

## 定义和用法

data() 方法向被选元素附加数据，或者从被选元素获取数据。

注释：这是底层级的方法；使用 .data() 更加方便。

## 从元素返回数据

从被选元素中返回附加的数据。

### 语法

```
$(_selector_).data(_name_)
```

| 参数 | 描述 |
| --- | --- |
| *name* | 可选。规定要取回的数据的名称。如果没有规定名称，则该方法将以对象的形式从元素中返回所有存储的数据。 |

## 向元素附加数据

向被选元素附加数据。

### 语法

```
$(_selector_).data(_name_,_value_)
```

| 参数 | 描述 |
| --- | --- |
| *name* | 必需。规定要设置的数据的名称。 |
| *value* | 必需。规定要设置的数据的值。 |

# 使用对象向元素附加数据

使用带有名称/值对的对象向被选元素添加数据。

## 语法

```
$(_selector_).data(_object_)
```

| 参数 | 描述 |
| --- | --- |
| *object* | 必需。规定包含名称/值对的对象。 |

# jQuery 遍历 - dequeue() 方法

## 实例

使用 dequeue() 终止一个自定义的队列函数：

```
$("div").queue(function () {
  $(this).toggleClass("red");
  $(this).dequeue();
});
```

## 定义和用法

dequeue() 方法为匹配元素执行序列中的下一个函数。

### 语法

```
.dequeue(_queueName_)
```

| 参数 | 描述 |
|---|---|
| *queueName* | 可选。字符串值，包含序列的名称。默认是 fx, 标准的效果序列。 |

### 详细说明

当调用 .dequeue() 时，会从序列中删除下一个函数，然后执行它。该函数反过来会（直接或间接地）引发对 .dequeue() 的调用，这样序列才能继续下去。

# jQuery 遍历 - jQuery.dequeue() 方法

## 实例

使用 dequeue() 终止一个自定义的队列函数：

```
$("div").queue(function () {
  $(this).toggleClass("red");
  $(this).dequeue();
});
```

## 定义和用法

dequeue() 方法为匹配元素执行序列中的下一个函数。

注释：这是底层级的方法；使用 .dequeue() 更加方便。

### 语法

```
.dequeue(_queueName_)
```

| 参数 | 描述 |
| --- | --- |
| *queueName* | 可选。字符串值，包含序列的名称。默认是 fx, 标准的效果序列。 |

### 详细说明

当调用 .dequeue() 时，会从序列中删除下一个函数，然后执行它。该函数反过来会（直接或间接地）引发对 .dequeue() 的调用，这样序列才能继续下去。

# jQuery 遍历 - hasData() 方法

## 实例

在元素上设置数据，然后查看 hasData 的结果：

```
$(function(){
  var $p = jQuery("p"), p = $p[0];
  $p.append(jQuery.hasData(p)+" "); /* false */
  jQuery.data(p, "testing", 123);
  $p.append(jQuery.hasData(p)+" "); /* true */
  jQuery.removeData(p, "testing");
  $p.append(jQuery.hasData(p)+" "); /* false */
});
```

## 定义和用法

hasData() 方法检测元素是否拥有与之相关的任何 jQuery 数据。

### 语法

```
jQuery.hasData(_element_)
```

| 参数 | 描述 |
| --- | --- |
| *element* | 可选。需要检查其数据的 DOM 元素。 |

### 详细说明

jQuery.hasData() 方法检测元素当前是否拥有通过使用 jQuery.data() 设置的任何值。如果没有数据与元素相关（根本不存在数据对象或者数据对象为空），则该方法返回 false；否则返回 true。

jQuery.hasData(*element*) 的主要优势是，在不存在数据对象的情况下，不会创建并将数据对象与元素进行关联。相反地，jQuery.data(*element*) 总是向调用者返回数据对象，如果之前数据对象不存在，则会创建它。

# jQuery 遍历 - queue() 方法

## 实例

显示队列的长度：

```
function showIt() {
  var n = div.queue("fx");
  $("span").text( n.length );
  setTimeout(showIt, 100);
}
```

## 定义和用法

queue() 方法显示或操作在匹配元素上执行的函数队列。

### 语法

```
.queue(_queueName_)
```

| 参数 | 描述 |
|---|---|
| *queueName* | 可选。字符串值，包含序列的名称。默认是 fx, 标准的效果序列。 |

## 操作队列

queue() 方法操作在匹配元素上执行的函数队列。

### 语法

```
.queue(_queueName_,_newQueue_)
```

| 参数 | 描述 |
|---|---|
| *queueName* | 可选。字符串值，包含序列的名称。默认是 fx, 标准的效果序列。 |

### 详细说明

每个元素均可拥有一到多个由 jQuery 添加的函数队列。在大多数应用程序中，只使用一个队列（名为 fx）。队列运行在元素上异步地调用动作序列，而不会终止程序执行。典型例子时调用元素上的多个动画方法。例如：

```
$('#foo').slideUp().fadeIn();
```

当这条语句执行时，元素会立即开始其滑动动画，但是淡入过渡被置于 fx 队列，只有当滑动过渡完成后才会被调用。

.queue() 方法允许我们直接对这个函数队列进行操作。调用带有回调函数的 .queue() 方法特别有用；它允许我们在队列末端放置一个新函数。

这个特性与动画方法提供回调函数类似，但是无需在动画执行时设置回调函数。

```
$('#foo').slideUp();
$('#foo').queue(function() {
  alert('Animation complete.');
  $(this).dequeue();
});
```

等价于：

```
$('#foo').slideUp(function() {
  alert('Animation complete.');
});
```

请注意，当通过 .queue() 添加函数时，我们应当确保最终调用了 .dequeue()，这样下一个排队的函数才能执行。

## 例子 1

对自定义函数进行队列操作：

```
$(document.body).click(function () {
  $("div").show("slow");
  $("div").animate({left:'+=200'},2000);
  $("div").queue(function () {
    $(this).addClass("newcolor");
    $(this).dequeue();
  });
  $("div").animate({left:'-=200'},500);
  $("div").queue(function () {
    $(this).removeClass("newcolor");
    $(this).dequeue();
  });
  $("div").slideUp();
});
```

## 例子 2

设置队列数组来删除队列：

```
$("#start").click(function () {
  $("div").show("slow");
  $("div").animate({left:'+=200'},5000);
  $("div").queue(function () {
    $(this).addClass("newcolor");
    $(this).dequeue();
  });
  $("div").animate({left:'-=200'},1500);
  $("div").queue(function () {
    $(this).removeClass("newcolor");
    $(this).dequeue();
  });
  $("div").slideUp();
});

$("#stop").click(function () {
  $("div").queue("fx", []);
  $("div").stop();
});
```

# jQuery 遍历 - jQuery.queue() 方法

## 实例

显示队列的长度：

```
function showIt() {
  var n = div.queue("fx");
  $("span").text( n.length );
  setTimeout(showIt, 100);
}
```

## 定义和用法

queue() 方法显示或操作在匹配元素上执行的函数队列。

注释：这是底层级的方法；使用 .queue() 更加方便。

### 语法

```
.queue(_queueName_)
```

| 参数 | 描述 |
| --- | --- |
| *queueName* | 可选。字符串值，包含序列的名称。默认是 fx, 标准的效果序列。 |

## 操作队列

queue() 方法操作在匹配元素上执行的函数队列。

### 语法

```
.queue(_queueName_,_newQueue_)
```

| 参数 | 描述 |
| --- | --- |
| *queueName* | 可选。字符串值，包含序列的名称。默认是 fx, 标准的效果序列。 |

### 详细说明

每个元素均可拥有一到多个由 jQuery 添加的函数队列。在大多数应用程序中，只使用一个队列（名为 fx）。队列运行在元素上异步地调用动作序列，而不会终止程序执行。典型例子时调用元素上的多个动画方法。例如：

```
$('#foo').slideUp().fadeIn();
```

当这条语句执行时，元素会立即开始其滑动动画，但是淡入过渡被置于 fx 队列，只有当滑动过渡完成后才会被调用。

.queue() 方法允许我们直接对这个函数队列进行操作。调用带有回调函数的 .queue() 方法特别有用；它允许我们在队列末端放置一个新函数。

这个特性与动画方法提供回调函数类似，但是无需在动画执行时设置回调函数。

```
$('#foo').slideUp();
$('#foo').queue(function() {
  alert('Animation complete.');
  $(this).dequeue();
});
```

等价于：

```
$('#foo').slideUp(function() {
  alert('Animation complete.');
});
```

请注意，当通过 .queue() 添加函数时，我们应当确保最终调用了 .dequeue()，这样下一个排队的函数才能执行。

## 例子 1

对自定义函数进行队列操作：

```
$(document.body).click(function () {
  $("div").show("slow");
  $("div").animate({left:'+=200'},2000);
  $("div").queue(function () {
    $(this).addClass("newcolor");
    $(this).dequeue();
  });
  $("div").animate({left:'-=200'},500);
  $("div").queue(function () {
    $(this).removeClass("newcolor");
    $(this).dequeue();
  });
  $("div").slideUp();
});
```

## 例子 2

设置队列数组来删除队列：

```
$("#start").click(function () {
  $("div").show("slow");
  $("div").animate({left:'+=200'},5000);
  $("div").queue(function () {
    $(this).addClass("newcolor");
    $(this).dequeue();
  });
  $("div").animate({left:'-=200'},1500);
  $("div").queue(function () {
    $(this).removeClass("newcolor");
    $(this).dequeue();
  });
  $("div").slideUp();
});

$("#stop").click(function () {
  $("div").queue("fx", []);
  $("div").stop();
});
```

# jQuery 数据 - removeData() 方法

## 实例

从元素中删除之前添加的数据：

```
$("#btn2").click(function(){
  $("div").removeData("greeting");
  alert("Greeting is: " + $("div").data("greeting"));
});
```

## 定义和用法

removeData() 方法删除之前通过 data() 方法设置的数据。

### 语法

```
$(_selector_).removeData(_name_)
```

| 参数 | 描述 |
|---|---|
| *name* | 可选。规定要删除的数据的名称。如果没有规定名称，该方法将从被选元素中删除所有已存储的数据。 |

# jQuery 数据 - jQuery.removeData() 方法

## 实例

从元素中删除之前添加的数据：

```
$("#btn2").click(function(){
  $("div").removeData("greeting");
  alert("Greeting is: " + $("div").data("greeting"));
});
```

## 定义和用法

removeData() 方法删除之前通过 data() 方法设置的数据。

注释：这是底层级的方法；使用 .removeData() 更加方便。

### 语法

```
$(_selector_).removeData(_name_)
```

| 参数 | 描述 |
| --- | --- |
| *name* | 可选。规定要删除的数据的名称。如果没有规定名称，该方法将从被选元素中删除所有已存储的数据。 |

# jQuery 参考手册 - DOM 元素方法

## jQuery DOM 元素方法

| 函数 | 描述 |
| --- | --- |
| .get() | 获得由选择器指定的 DOM 元素。 |
| .index() | 返回指定元素相对于其他指定元素的 index 位置。 |
| .size() | 返回被 jQuery 选择器匹配的元素的数量。 |
| .toArray() | 以数组的形式返回 jQuery 选择器匹配的元素。 |

# jQuery DOM 元素方法 - get() 方法

## 实例

获得第一个 p 元素的名称和值：

```
$("button").click(function(){
  x=$("p").get(0);
  $("div").text(x.nodeName + ": " + x.innerHTML);
});
```

## 定义和用法

get() 方法获得由选择器指定的 DOM 元素。

### 语法

```
$(_selector_).get(_index_)
```

| 参数 | 描述 |
| --- | --- |
| *index* | 可选。规定获取哪个匹配元素（通过 index 编号）。 |

# jQuery DOM 元素方法 - index() 方法

## 实例

获得第一个 p 元素的名称和值：

```
$("li").click(function(){
  alert($(this).index());
});
```

## 定义和用法

index() 方法返回指定元素相对于其他指定元素的 index 位置。

这些元素可通过 jQuery 选择器或 DOM 元素来指定。

注释：如果未找到元素，index() 将返回 -1。

## 第一个匹配元素的 index，相对于同胞元素

获得第一个匹配元素相对于其同胞元素的 index 位置。

### 语法

```
$(_selector_).index()
```

## 元素的 index，相对于选择器

获得元素相对于选择器的 index 位置。

该元素可以通过 DOM 元素或 jQuery 选择器来指定。

### 语法

```
$(_selector_).index(_element_)
```

| 参数 | 描述 |
| --- | --- |
| *element* | 可选。规定要获得 index 位置的元素。可以是 DOM 元素或 jQuery 选择器。 |

# jQuery DOM 元素方法 - size() 方法

## 实例

输出被 jQuery 选择器匹配的元素的数量：

```
$("button").click(function(){
  alert($("li").size());
});
```

## 定义和用法

size() 方法返回被 jQuery 选择器匹配的元素的数量。

### 语法

```
$(_selector_).size()
```

# jQuery DOM 元素方法 - toArray() 方法

## 实例

将 li 元素转换为数组，然后输出该数组元素的 innerHTML ：

```
$("button").click(function(){
  x=$("li").toArray()
  for (i=0;i<x.length;i++)
    {
    alert(x[i].innerHTML);
    }
});
```

## 定义和用法

toArray() 方法以数组的形式返回 jQuery 选择器匹配的元素。

### 语法

```
$(_selector_).toArray()
```

# jQuery 参考手册 - 核心

## jQuery 核心函数

| 函数 | 描述 |
| --- | --- |
| jQuery() | 接受一个字符串，其中包含了用于匹配元素集合的 CSS 选择器。 |
| jQuery.noConflict() | 运行这个函数将变量 $ 的控制权让渡给第一个实现它的那个库。 |

# jQuery 核心 - jQuery() 方法

## 实例

找出所有属于 div 元素的子元素的 p 元素，然后设置其边框属性：

```
$("div > p").css("border", "1px solid gray");
```

## 定义和用法

jQuery() 方法接受一个字符串，其中包含了用于匹配元素集合的 CSS 选择器。

jQuery() 函数有三种语法：

### 语法 1

接受一个字符串，其中包含了用于匹配元素集合的 CSS 选择器：

```
jQuery(_selector_, [_context_])
```

详细用法

### 语法 2

使用原始 HTML 的字符串来创建 DOM 元素：

```
jQuery(_html_,[_ownerDocument_])
```

详细用法

### 语法 3

绑定一个在 DOM 文档载入完成后执行的函数：

```
jQuery( callback )
```

详细用法

# jQuery( selector, [ context ] )

该语法有以下几种用法：

## 用法 1 ：设置选择器环境

### 语法

```
jQuery(_selector_, [_context_])
```

默认情况下，选择器从文档根部对 DOM 进行搜索。不过，可以为 $() 设置可选的 context 参数。

例如，如果我们希望在一个 callback 中搜索一个元素，可以限定下面的搜索：

### 实例

```
$("div.foo").click(function() {
  $("span", this).addClass("bar");
});
```

由于我们已经将 span 选择器限定到 this 这个环境中，只有被点击元素中的 span 会得到附加的 class。

在内部，选择器环境是通过 .find() 方法实现的，因此 $("span", this) 等价于 $(this).find("span")。

jQuery 的核心功能都是通过这个函数实现的。jQuery中的一切都基于这个函数，或者说都是在以某种方式使用这个函数。这个函数最基本的用法就是向它传递一个表达式（通常由 CSS 选择器组成），然后根据这个表达式来查找所有匹配的元素。

默认情况下, 如果没有指定 context 参数，$() 将在当前的 HTML document 中查找 DOM 元素；如果指定了 context 参数，如一个 DOM 元素集或 jQuery 对象，那就会在这个 context 中查找。在 jQuery 1.3.2 以后，其返回的元素顺序等同于在 context 中出现的先后顺序。

## 用法 2 ：使用 DOM 元素

### 语法

```
jQuery(_element_)
```

该函数允许我们通过使用以其他方式找到的 DOM 元素来创建 jQuery 对象。该功能通常的用法是，对已经通过 this 关键字传递到 callback 函数的元素调用 jQuery 的方法：

实例

```
$("div.foo").click(function() {
  $(this).slideUp();
});
```

此例会在元素被点击时使用滑动动画对其进行隐藏。由于处理程序接受的 this 关键词中的被点击项目是纯的 DOM 元素，因此在对其调用 jQuery 的方法之前，必须用 jQuery 对象包装该元素。

这个函数也可以接收 XML 文档和 Window 对象（虽然它们不是 DOM 元素）作为有效的参数。

当 XML 数据从 Ajax 调用中返回后，我们可以使用 $() 函数通过 jQuery 对象包装该数据。一旦完成，我们就可以使用 .find() 和其他 DOM 遍历方法来取回 XML 结构中单个元素。

## 用法 3 ：克隆 jQuery 对象

语法

```
jQuery(_jQuery object_)
```

当以参数的形式向 $() 函数传递 jQuery 对象后，会创建一个该对象的副本。与初始对象一样，新的 jQuery 对象引用相同的 DOM 元素。

## 用法 4 ：返回空的集合

语法

```
jQuery()
```

对于 jQuery 1.4，调用无参数的 jQuery() 方法会返回空的 jQuery 集合。在之前版本的 jQuery 中，这样会返回包含 document 节点的集合。

# jQuery( html, [ ownerDocument ] )

该语法有以下几种用法：

## 用法 1 ：创建新的元素

语法

```
jQuery(_html_,[_ownerDocument_])
```

你可以传递一个手写的 HTML 字符串，或者由某些模板引擎或插件创建的字符串，也可以是通过 AJAX 加载过来的字符串。但是在你创建 input 元素的时会有限制，可以参考第二个示例。

当然这个字符串可以包含斜杠 (比如一个图像地址)，还有反斜杠。当你创建单个元素时，请使用闭合标签或 XHTML 格式。例如，创建一个 span ，可以用 $("<span/>") 或 $("<span></span>") ，但不推荐 $("<span>")。在 jQuery 中，这个语法等同于 $(document.createElement("span"))。

如果以参数的形式将字符串传递给 $()，jQuery 会检查字符串是否是 HTML （比如，字符串某些位置存在标签）。如果不是，则把字符串解释为选择器表达式，请见上面的讲解。但如果字符串是 HTML 片段，则 jQuery 试图创建由该 HTML 片段描述的 DOM 元素。然后会创建并返回一个引用这些 DOM 元素的 jQuery 对象：

### 实例

```
$("<p id="test">My <em>new</em> text</p>").appendTo("body");
```

如果 HTML 片段比不含属性的简单标签更复杂，如同上面例子中的 HTML，那么元素实际的创建过程是由浏览器的 innerHTML 机制完成的。具体地讲，jQuery 会创建新的 <div> 元素，然后为传入的 HTML 片段设置元素的 innerHTML 属性。当参数只是简单的标签，比如$("<img />") 或 $("<a></a>")，jQuery 会通过内生的 JavaScript createElement() 函数来创建元素。

要确保跨平台兼容性，片段的结构必须良好。能够包含其他元素的标签必须成对出现（带有关闭标签）：

```
$("<a href="http://jquery.com"></a>");
```

不过，jQuery 也允许类似 XML 的标签语法：

```
$("<a/>");
```

无法包含其他元素的标签可以关闭，也可以不关闭：

```
$("<img />");
$("<input>");
```

## 用法 2 ：设置属性和事件

### 语法

```
jQuery(_html_,_props_)
```

对于 jQuery 1.4，我们可以向第二个参数传递一个属性映射。该参数接受能够传递给 .attr() 方法的属性的超集。此外，可以传递任意的事件类型，并可以调用下面的 jQuery 方法：val, css, html, text, data, width, height, or offset.

注意，Internet Explorer 不允许你创建 input 元素并改变其类型；您必须使用例如 "<input type="checkbox" />" 来规定类型。

### 实例

创建一个 <input> 元素，同时设定 type 属性、属性值，以及一些事件。

```
$("<input>", {
  type: "text",
  val: "Test",
  focusin: function() {
    $(this).addClass("active");
  },
  focusout: function() {
    $(this).removeClass("active");
  }
}).appendTo("form");
```

# jQuery( callback )

允许你绑定一个在 DOM 文档载入完成后执行的函数。

该函数的作用如同 $(document).ready() 一样，只不过用这个函数时，需要把页面中所有需要在 DOM 加载完成时执行的其他 $() 操作符都包装到其中来。尽管从技术上来说，这个函数是可链接的，但真正以这种方式链接的情况并不多。

## 例子

当DOM加载完成后，执行其中的函数：

```
$(function(){
  // 文档就绪
});
```

# jQuery 核心 - noConflict() 方法

## 实例

使用 noConflict() 方法为 jQuery 变量规定新的名称：

```
var jq=$.noConflict();
```

## 定义和用法

noConflict() 方法让渡变量 $ 的 jQuery 控制权。

该方法释放 jQuery 对 $ 变量的控制。

该方法也可用于为 jQuery 变量规定新的自定义名称。

提示：在其他 JavaScript 库为其函数使用 $ 时，该方法很有用。

### 语法

```
jQuery.noConflict(_removeAll_)
```

| 参数 | 描述 |
|---|---|
| *removeAll* | 布尔值。指示是否允许彻底将 jQuery 变量还原。 |

### 说明

许多 JavaScript 库使用 $ 作为函数或变量名，jQuery 也一样。在 jQuery 中，$ 仅仅是 jQuery 的别名，因此即使不使用 $ 也能保证所有功能性。假如我们需要使用 jQuery 之外的另一 JavaScript 库，我们可以通过调用 $.noConflict() 向该库返回控制权：

```
<script type="text/javascript" src="other_lib.js"></script>
<script type="text/javascript" src="jquery.js"></script>

<script type="text/javascript">
  $.noConflict();
  // 使用另一个库的 $ 的代码
</script>
```

可以与 .ready() 方法结合起来使用，来为 jQuery 对象起别名，这项技术非常有效：

```
<script type="text/javascript" src="other_lib.js"></script>
<script type="text/javascript" src="jquery.js"></script>

<script type="text/javascript">
  $.noConflict();
  jQuery(document).ready(function($) {
    // 使用 jQuery $ 的代码
  });
  // 使用其他库的 $ 的代码
</script>
```

此外，通过向该方法传递参数 true，我们可以将 $ 和 jQuery 的控制权都交还给原来的库。用之前请考虑清楚！

这是相对于简单的 noConflict 方法更极端的版本，因为这将完全重新定义 jQuery。这通常用于一种极端的情况，比如你想要将 jQuery 嵌入一个高度冲突的环境。注意：调用此方法后极有可能导致插件失效。

# 实例

## 例子 1

将 $ 引用的对象映射回原始的对象：

```
jQuery.noConflict();

jQuery("div p").hide();    // 使用 jQuery

$("content").style.display = "none";    // 使用其他库的 $()
```

## 例子 2

恢复使用别名 $，然后创建并执行一个函数，在这个函数的作用域中仍然将 $ 作为 jQuery 的别名来使用。在这个函数中，原来的 $ 对象是无效的。这个函数对于大多数不依赖于其他库的插件都十分有效：

```
jQuery.noConflict();

(function($) {
  $(function() {
    // 使用 $ 作为 jQuery 别名的代码
  });
})(jQuery);

... // 其他用 $ 作为别名的库的代码
```

## 例子 3

可以将 jQuery.noConflict() 与简写的 ready 结合，使代码更紧凑：

```
jQuery.noConflict()(function(){
    // 使用 jQuery 的代码
});

... // 其他库使用 $ 做别名的代码
```

# 例子 4

创建一个新的别名用以在接下来的库中使用 jQuery 对象：

```
var j = jQuery.noConflict();

j("div p").hide();    // 基于 jQuery 的代码

$("content").style.display = "none";    // 基于其他库的 $() 代码
```

# 例子 5

完全将 jQuery 移到一个新的命名空间：

```
var dom = {};
dom.query = jQuery.noConflict(true);
```

结果:

```
dom.query("div p").hide();    // 新 jQuery 的代码

$("content").style.display = "none";    // 另一个库 $() 的代码

jQuery("div > p").hide();    // 另一个版本 jQuery 的代码
```

# jQuery 参考手册 - 属性

## jQuery 属性

下面列出的这些方法设置或返回元素的 CSS 相关属性。

| 属性 | 描述 |
| --- | --- |
| context | 在版本 1.10 中被弃用。包含传递给 jQuery() 的原始上下文。 |
| jquery | 包含 jQuery 版本号。 |
| jQuery.fx.interval | 改变以毫秒计的动画速率。 |
| jQuery.fx.off | 全局禁用/启用所有动画。 |
| jQuery.support | 表示不同浏览器特性或漏洞的属性集合（用于 jQuery 内部使用）。 |
| length | 包含 jQuery 对象中的元素数目。 |

# jQuery context 属性

## 实例

检测上下文：

```
$("div").append("<p>" + $("div").context + "</p>")
.append("<p>" + $("div",document.body).context.nodeName + "</p>");
```

## 定义和用法

context 属性在 jQuery version 1.10 中被弃用。

context 属性含有被传递到 jQuery 的原始上下文，可能是 DOM 节点上下文，如果未传递节点，则是 document 上下文。

### 语法

```
context
```

# jQuery jquery 属性

## 实例

输出当前正在运行的 jQuery 版本：

```
$("button").on("click",function(){
  var version = $().jquery;
  alert("You are running jQuery version: " + version);
});
```

## 定义和用法

jquery 属性返回的字符串包含 jquery 的版本号。

### 语法

```
$().jquery
```

# jQuery jQuery.fx.interval 属性

## 实例

以较少的帧数来运行 <div> 元素的动画：

```
$("#toggle").on("click",function(){
  $("div").toggle(5000);
});
$("#interval").on("click",function(){
  jQuery.fx.interval = 500;
});
```

## 定义和用法

jQuery.fx.interval 属性用于改变以毫秒计的动画运行速率。可操作该属性来调整动画运行的每秒帧数。

默认值是 13 毫秒。该属性常用于修改动画运行的每秒帧数。

降低这个值能够使动画在更快的浏览器中运行得更流畅，但这么做也行会影响性能。

提示：由于 jQuery 使用一个全局的间隔时间，为了使该属性生效，动画应该不在运行或者首先停止所有动画。

注释：该属性在支持 requestAnimationFrame 属性的浏览器中无效，比如 Google Chrome 11。

### 语法

```
jQuery.fx.interval = _milliseconds_;
```

| 属性 | 描述 |
| --- | --- |
| *milliseconds* | 必需。规定以毫秒计的动画运行速率。默认是 13 毫秒。 |

# jQuery jQuery.fx.off 属性

## 实例

切换动画开关：

```
$("#true").click(function(){
  jQuery.fx.off = true;
});
$("#false").click(function(){
  jQuery.fx.off = false;
});
$("#toggle").click(function(){
  $("div").toggle("slow");
});
```

## 定义和用法

jQuery.fx.off 属性用于对所有动画进行全局禁用或启用。

默认值是 false，它允许动画正常运行。当设置为 true 时，将禁用所有动画方法，这样会把元素设置为其最后的状态，而不是显示效果。

提示：如需简化代码，可以使用 $.fx.off 来代替 jQuery.fx.off。

### 语法

```
jQuery.fx.off = true|false;
```

| 属性 | 描述 |
| --- | --- |
| true | 规定应该禁用动画。 |
| false | 默认。规定应该启用动画。 |

# jQuery jQuery.support 属性

## 实例

测试浏览器是否能创建 XMLHttpRequest 对象：

```
$(document).ready(function(){
  $("p").html("This browser can create XMLHttpRequest object: " + jQuery.support.ajax);
});
```

## 定义和用法

jQuery.support 属性包含表示不同浏览器特性或漏洞的属性集。

此属性主要用于 jQuery 的内部使用。

### 语法

```
jQuery.support._propvalue_
```

| 属性 | 描述 |
| --- | --- |
| *propvalue* | 必需。规定要测试的功能。这些测试包括：`ajax` `boxModel` `changeBubbles` `checkClone` `checkOn` `cors` `cssFloat` `hrefNormalized` `htmlSerialize` `leadingWhitespace` `noCloneChecked` `noCloneEvent` `opacity` `optDisabled` `optSelected` `scriptEval()` `style` `submitBubbles` `tbody` |

# jQuery length 属性

## 实例

输出 <li> 元素的数目：

```
$("button").click(function(){
  alert($("li").length);
});
```

## 定义和用法

length 属性包含 jQuery 对象中元素的数目。

### 语法

```
$(selector).length
```

# 免责声明

W3School提供的内容仅用于培训。我们不保证内容的正确性。通过使用本站内容随之而来的风险与本站无关。W3School简体中文版的所有内容仅供测试，对任何法律问题及风险不承担任何责任。